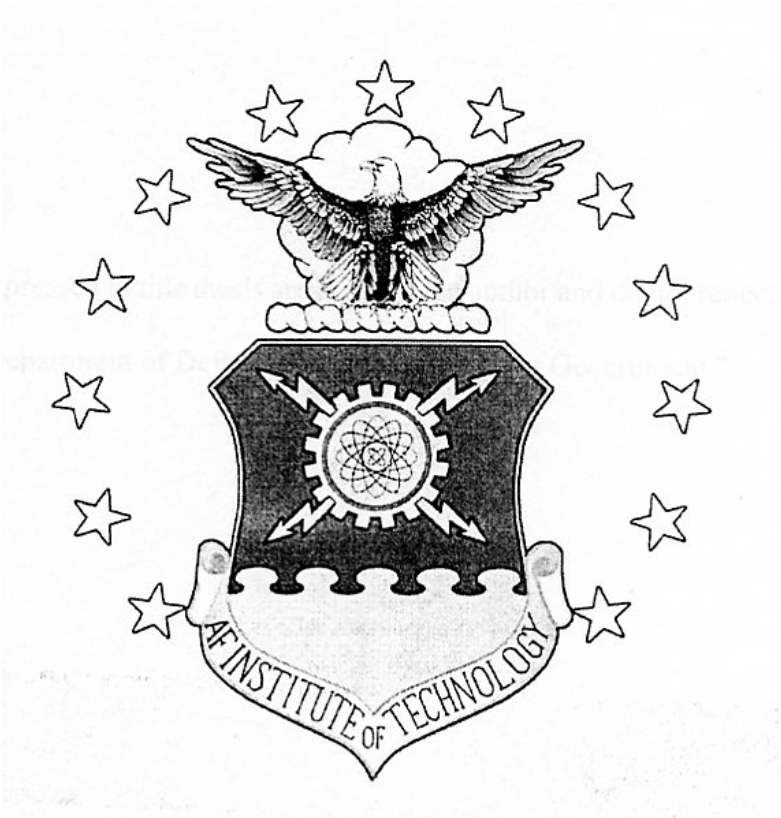

AN INVESTIGATION INTO ROBUST WIND CORRECTION ALGORITHMS FOR OFF-THE-SHELF UNMANNED AERIAL VEHICLE AUTOPILOTS

THESIS

Brent K. Robinson, Ensign, USN
AFIT/GAE/ENY/06-J14

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GAE/ENY/06-J14

AN INVESTIGATION INTO ROBUST WIND CORRECTION ALGORITHMS FOR OFF-THE-SHELF UNMANNED AERIAL VEHICLE AUTOPILOTS

THESIS

Presented to the Faculty

Department of Aeronautics and Astronautics

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Aeronautical Engineering

Brent K. Robinson, BS

Ensign, USN

June 2006

AFIT/GAE/ENY/06-J14

ROBUST WIND CORRECTION ALGORITHM FOR OFF-THE-SHELF UNMANNED AERIAL VEHICLE AUTOPILOTS

Brent K. Robinson, BS
Ensign, USN

Approved:

_____	_____
Paul Blue, Major, USAF (Chairman)	date
_____	_____
Dr. John F. Racquet (Member)	date
_____	_____
Dr. David R. Jacques (Member)	date

Abstract

This research effort focuses on developing methods to design efficient wind correction algorithms to “piggy-back” on current off-the-shelf Unmanned Aerial Vehicle (UAV) autopilots. Autonomous flight is certainly the near future for the aerospace industry and there exists great interest in defining a system that can guide and control small aircraft with high levels of accuracy. The primary systems required to command the vehicles are already in place, but with only moderate abilities to adjust for dynamic environments (i.e., wind effects), if at all. The goal of this research is to develop a systematic procedure for implementing efficient and robust wind effects corrections to existing autopilots used on small Unmanned Aerial Vehicles. The research will investigate the feasibility of an external dynamic environment control algorithm as a means of improving current, off-the-shelf autopilot technology relating to small UAVs. The research then presents three main focuses. First, a determination of the estimated winds utilizing the existing, on-board sensors. Second, the development of a wind correction algorithm that incorporates simple mathematical principals to counter the 2-Dimensional wind forces acting on the aircraft; and third, the integration of that wind compensator into the on-board navigational system. This “piggy-back” algorithm must assimilate smoothly with the current GPS technologies to provide acceptable and safe flight path following. The design procedures developed were demonstrated in simulation and with flight tests on the SIG Rascal 110 UAV. This report builds the framework from which current wind correction research at AFIT and the ANT Center is based.

Acknowledgements

I would like to express my sincerest appreciation to my faculty advisor, Major Paul Blue, for his guidance and insight. His commitment to the success of his students and their research provided a superb example of leadership from which I will attempt to emulate throughout my career. I would like to thank Dr. John Raquet for running and maintaining the professional and educational atmosphere of AFIT's Advanced Navigation Laboratory, without which the research would not have been possible. I must also acknowledge a few others whose work was critical to the success of this project. First to Athan Waldron, he put in the numerous hours of hands on labor required to continually supply the ANT lab with the aircraft and all their components. His meticulous understanding of the aircraft and the avionics allowed for a smooth integration into the program. John McNees, our radio control aircraft expert and pilot, guided the aircraft through any flight testing with unparalleled experience and wisdom. Don Smith, the lab's expert technician in everything mechanical and electrical. Don was always there to answer the unanswerable and connect all the loose ends. Randy Plate, who provided the initial work on Piccolo's Software Development Kit. Steve Rasmussen, from the Air Force Research Labs, provided timely and much needed expertise in C++ programming. To Second Lieutenant Brett Pagel, also from the Air Force Research Labs, who aided with previous autonomous UAV experience. Finally, to my peers in the lab who could always be counted on to lighten the mood or provide a necessary distraction.

Brent K. Robinson

Table of Contents

Page

Abstract iv
Table of Contents vi
List of Figures viii
List of Tables xiii
List of Tables xiii
I. Introduction 1
1.1 – Motivation 1
1.2 – Problem Statement 4
1.3 – Research Objectives 5
1.4 – Significance of Research 5
1.5 – Methodology 6
1.6 – Thesis Preview 8
II. Background 9
2.1 – Overview 9
2.2 – Aircraft 9
2.2.1 – Airframe 9
2.2.2 – Engine and Propeller 12
2.3 – Avionics 14
2.3.1 – Radio Control System 14
2.3.2 – Piccolo II Autopilot 15
2.3.3 – Honeywell HMR2300 Digital Magnetometer 20
2.4 – Simulation 21
2.4.1 – Hardware in the Loop (HITL) 22
2.4.2 – Software Development Kit (SDK) 23
2.5 – Flight Testing 24
2.5.1 – Overview of Flight Test 24
2.5.2 – Flight Test Range 25
2.5.3 – Ground Equipment 27
2.5.4 – Criteria for Flight Test of UAVs at WPAFB 28
2.5.5 – Wind Correction Implementation 29
2.5.6 – Data Collection and Handling 29
2.6 – Chapter Summary 31

Page

III. Development of the Wind Correction Approaches 33

3.1 – Overview 33
3.2 – Real Time Wind Estimating 33
3.3 – Turn Rate Approach Equations 35
3.4 – Updating "Rabbit" Waypoint Approach 38
3.5 – Wind Corrected Sensor Pointing 40
3.6 – Wind Correction Implementation 43
3.6.1 – Real Time Wind Estimating 43
3.6.2 – Implementing the Turn Rate & Updating "Rabbit" Waypoint 45
3.6.3 – Wind Corrected Sensor Pointing 50
3.7 – Chapter Summary 53

IV. HITL Test Results and Analysis 54

4.1 – Overview 54
4.2 – Standard HITL Simulated Flight Tests with Real Time Wind Estimating 54
4.3 – HITL Simulation with Wind Correction 71
4.3.1 – Turn Rate & Updating "Rabbit" Waypoint Approaches 71
4.3.2 – Wind Corrected Sensor Pointing 72
4.4 –Flight Testing with Wind Correction 80
4.4.1 – Real Time Wind Estimating 80
4.4.2 – Turn Rate & Updating "Rabbit" Waypoint Approaches 80
4.4.3 – Wind Corrected Sensor Pointing 80
4.5 – Chapter Conclusions 81

V. Conclusions and Recommendations 82

5.1 – Conclusions 82
5.2 – Recommendations 84

Appendix A: Complete Set of Simulated Test Results 86

Appendix B: Software Development Kit (SDK) C++ Code 124

Appendix C: MATLAB Code 145

Appendix D: Proposed Actual Flight Test Plans 156

Appendix E: Flight Test Results 162

Bibliography 165

Vita 167

List of Figures

Page

Figure 1. Two Completed Sig Rascal 110's (Jodeh, 2006) 10

Figure 2. Sig Rascal Wing Planform View (Jodeh, 2006) 11

Figure 3. O.S. FS-120S III Four Cycle Engine 13

Figure 4. APC 16x8 Nylon Propeller 13

Figure 5. Futaba 9CAP/9CAF 8 Channel Transmitter 14

Figure 6. Piccolo II Block Diagram of Internal Components 16

Figure 7. Piccolo II Airborne Autopilot Unit 17

Figure 8. Required Ground Equipment (minus the laptops) for the Piccolo II Autopilot System 18

Figure 9. Fail Safe Control Relay Schematic 19

Figure 10. Complete Autonomous Flight Setup 20

Figure 11. Honeywell HMR2300 Digital Magnetometer (Honeywell, 2004) 21

Figure 12. Standard Hardware in the Loop Simulation Setup 22

Figure 13. WPAFB, Area B Flight Test Range 26

Figure 14. Ground Equipment and Test Team Conducting a Flight Test 28

Figure 15. Top View of the UAV with the Adjustment Parameters Defined 42

Figure 16. Screen Capture of the Piccolo SDK Executable 44

Figure 17. Standard UAV & Sensor Tracks for a Point-to-Point Flight Path 55

Figure 18. Various Flight Characteristics for the Standard Point-to-Point Flight. 56

Figure 19. Wind Estimations & Cross Track Distance. 57

Figure 20. Circular Orbit Flight Path with Constant Velocity and Wind 59

Figure 21. Various Parameters of the Circular Orbit Flight Path 59

Figure 22. Estimated Wind Values for the Circular Orbit 60

Figure 23. Race Track Pattern with TAS=12m/s & Wind= 5m/s 62

Figure 24. Various Parameters for the Race Track Pattern at 12m/s and TC=250 62

Figure 25. Wind Estimations & Cross Track Distance 63

Page

Figure 26. Race Track Pattern at 20 m/s Track Conv.=250 64
Figure 27. Race Track Pattern at 30m/s with Track Conv.=250 65
Figure 28. Race Track Pattern at 12 m/s with Track Conv.=150 67
Figure 29. Race Track Pattern at 12 m/s with Track Conv.=50 68
Figure 30. Race Track Pattern at 20 m/s with Track Conv.=150 69
Figure 31. Race Track Pattern at 20 m/s with Track Conv.=50 70
Figure 32. Point to Point at 20 m/s - Adjusted for Sensor 73
Figure 33. Race Track Pattern at 12 m/s - Adjusted Waypoints 74
Figure 34. Race Track Pattern at 15 m/s - Adjusted Waypoints 76
Figure 35. Race Track Pattern at 20 m/s - Adjusted Waypoints 77
Figure 36. Race Track Pattern at 30 m/s - Adjusted Waypoints 77
Figure 37. Point to Point at 20 m/s and 20% Lower Altitude 78
Figure 38. Point to Point at 20 m/s with 10 m/s Wind from the North 79
Figure 39. Standard UAV Short Point to Point at 12 m/s with Wind=5 m/s 86
Figure 40. Various Parameters for Short Point to Point at 12 m/s 86
Figure 41. Real Time Wind Estimations for Short Point to Point at 12 m/s 87
Figure 42. Standard UAV Short Point to Point at 15 m/s with Wind=5 m/s 87
Figure 43. Various Parameters for Short Point to Point at 15 m/s 88
Figure 44. Real Time Wind Estimations for Short Point to Point at 15 m/s 88
Figure 45. Standard UAV Short Point to Point at 20 m/s with Wind=5 m/s 89
Figure 46. Various Parameters for Short Point to Point at 20 m/s 89
Figure 47. Real Time Wind Estimations for Short Point to Point at 20 m/s 90
Figure 48. Standard UAV Short Point to Point at 30 m/s with Wind=5 m/s 90
Figure 49. Various Parameters for Short Point to Point at 30 m/s 91
Figure 50. Real Time Wind Estimations for Short Point to Point at 30 m/s 91

Page

Figure 51. Standard UAV Circular Orbit at 20 m/s 92

Figure 52. Various Parameters for the Circular Orbit at 20 m/s 92

Figure 53. Real Time Wind Estimations for the Circular Orbit at 20 m/s 93

Figure 54. Standard UAV Race Track Pattern at 12 m/s with Wind=5 m/s and TC=250 94

Figure 55. Various Parameters for the Race Track Pattern at 12 m/s, Wind5 m/s, & TC=250 94

Figure 56. Real Time Wind Estimations for the Race Track at 12 m/s, Wind=5 m/s, & TC=250 95

Figure 57. Standard UAV Race Track Pattern at 15 m/s with Wind=5 m/s and TC=250 95

Figure 58. Various Parameters for the Race Track Pattern at 15 m/s, Wind5 m/s, & TC=250 96

Figure 59. Real Time Wind Estimations for the Race Track at 15 m/s, Wind=5 m/s, & TC=250 96

Figure 60. Standard UAV Race Track Pattern at 20 m/s with Wind=5 m/s and TC=250 97

Figure 61. Various Parameters for the Race Track Pattern at 20 m/s, Wind5 m/s, & TC=250 97

Figure 62. Real Time Wind Estimations for the Race Track at 20 m/s, Wind=5 m/s, & TC=250 98

Figure 63. Standard UAV Race Track Pattern at 30 m/s with Wind=5 m/s and TC=250 98

Figure 64. Various Parameters for the Race Track Pattern at 30 m/s, Wind5 m/s, & TC=250 99

Figure 65. Real Time Wind Estimations for the Race Track at 30 m/s, Wind=5 m/s, & TC=250 99

Figure 66. Standard UAV Race Track Pattern at 12 m/s with Wind=5 m/s and TC=150 100

Figure 67. Various Parameters for the Race Track Pattern at 12 m/s, Wind5 m/s, & TC=150 100

Figure 68. Real Time Wind Estimations for the Race Track at 12 m/s, Wind=5 m/s, & TC=150 101

Figure 69. Standard UAV Race Track Pattern at 15 m/s with Wind=5 m/s and TC=150 101

Figure 70. Various Parameters for the Race Track Pattern at 15 m/s, Wind5 m/s, & TC=150 102

Figure 71. Real Time Wind Estimations for the Race Track at 15 m/s, Wind=5 m/s, & TC=150 102

Figure 72. Standard UAV Race Track Pattern at 20 m/s with Wind=5 m/s and TC=150 103

Figure 73. Various Parameters for the Race Track Pattern at 20 m/s, Wind5 m/s, & TC=150 103

Figure 74. Real Time Wind Estimations for the Race Track at 20 m/s, Wind=5 m/s, & TC=150 104

Figure 75. Standard UAV Race Track Pattern at 30 m/s with Wind=5 m/s and TC=150 104

Page

Figure 76. Various Parameters for the Race Track Pattern at 30 m/s, Wind5 m/s, & TC=150 105

Figure 77. Real Time Wind Estimations for the Race Track at 30 m/s, Wind=5 m/s, & TC=150 105

Figure 78. Standard UAV Race Track Pattern at 12 m/s with Wind=5 m/s and TC=50 106

Figure 79. Various Parameters for the Race Track Pattern at 12 m/s, Wind5 m/s, & TC=50 106

Figure 80. Real Time Wind Estimations for the Race Track at 12 m/s, Wind=5 m/s, & TC=50 107

Figure 81. Standard UAV Race Track Pattern at 15 m/s with Wind=5 m/s and TC=50 107

Figure 82. Various Parameters for the Race Track Pattern at 15 m/s, Wind5 m/s, & TC=50 108

Figure 83. Real Time Wind Estimations for the Race Track at 15 m/s, Wind=5 m/s, & TC=50 108

Figure 84. Standard UAV Race Track Pattern at 20 m/s with Wind=5 m/s and TC=50 109

Figure 85. Various Parameters for the Race Track Pattern at 20 m/s, Wind5 m/s, & TC=50 109

Figure 86. Real Time Wind Estimations for the Race Track at 20 m/s, Wind=5 m/s, & TC=50 110

Figure 87. Standard UAV Race Track Pattern at 30 m/s with Wind=5 m/s and TC=50 110

Figure 88. Various Parameters for the Race Track Pattern at 30 m/s, Wind5 m/s, & TC=50 111

Figure 89. Real Time Wind Estimations for the Race Track at 30 m/s, Wind=5 m/s, & TC=50 111

Figure 90. Updated UAV Race Track Pattern at 12 m/s with Wind=5 m/s and TC=250..... 112

Figure 91. Various Parameters for the Race Track Pattern at 12 m/s, Wind5 m/s, & TC=250 112

Figure 92. Real Time Wind Estimations for the Race Track at 12 m/s, Wind=5 m/s, & TC=250 113

Figure 93. Updated UAV Race Track Pattern at 15 m/s with Wind=5 m/s and TC=250..... 113

Figure 94. Various Parameters for the Race Track Pattern at 15 m/s, Wind5 m/s, & TC=250 114

Figure 95. Real Time Wind Estimations for the Race Track at 15 m/s, Wind=5 m/s, & TC=250 114

Figure 96. Updated UAV Race Track Pattern at 20 m/s with Wind=5 m/s and TC=250..... 115

Figure 97. Various Parameters for the Race Track Pattern at 20 m/s, Wind5 m/s, & TC=250 115

Figure 98. Real Time Wind Estimations for the Race Track at 20 m/s, Wind=5 m/s, & TC=250 116

Figure 99. Updated UAV Race Track Pattern at 30 m/s with Wind=5 m/s and TC=250..... 116

Figure 100. Various Parameters for the Race Track Pattern at 30 m/s, Wind5 m/s, & TC=250 117

Page

Figure 101. Real Time Wind Estimations for the Race Track at 30 m/s, Wind=5 m/s, & TC=250 117

Figure 102. Updated Long Point to Point at 20 m/s with Wind=5 m/s and TC=250..... 118

Figure 103. Various Parameters for the Long Point to Point at 20 m/s, Wind5 m/s, & TC=250 118

Figure 104. Real Time Wind Estimations for the Point to Point at 20 m/s, Wind=5 m/s, & TC=250 119

Figure 105. Updated Long Point to Point at 20 m/s with Wind=5 m/s & Lower Alt 120

Figure 106. Various Parameters for the Long Point to Point at 20 m/s, Wind5 m/s, & Lower Alt 120

Figure 107. Real Time Wind Estimations for the Point to Point at 20 m/s, Wind=5 m/s, & Lower Alt 121

Figure 108. Updated UAV for Point to Point with Wind =10 from North 122

Figure 109. Various Parameters for the Point to Point with Wind=10 m/s from the North 122

Figure 110. Real Time Wind Estimations for the Point to Point with the Wind=10 m/s from the North... 123

List of Tables

Page

Table 1. Various Sig Rascal 110 Characteristics 12

Table 2. Prominent Criteria for Flight Tests (Jodeh 2006) 29

Table 3. Available Telemetry through the Piccolo SDK 31

ROBUST WIND CORRECTION ALGORITHM FOR OFF-THE-SHELF UNMANNED AERIAL VEHICLE AUTOPILOTS

I. Introduction

1.1 – Motivation

The first one hundred years of flight brought about an incredible evolution beginning with two, small town bicycle makers soaring just over 120 feet and progressing to the global military and civil aerospace business of current times. This transformation has thrust aviation into the forefront of the world's daily operations and has positioned the business as a necessity in the everyday world. While this "revolution" has been rapid in historic terms and some have declared Aerospace as a mature business/technology, the next one hundred years will undoubtedly bring a myriad of advances that will continue to change how the world lives and operates. One of the most important developments of current times is that of Unmanned Aerial Vehicles (UAVs). While they have been envisaged as long as manned aircraft, the enabling technologies have only recently matured enough to bring them to a state of operational reality. Thus, UAVs of all sizes and capabilities are beginning to accomplish numerous missions impractical, or even impossible, for manned aircraft.

Leading the drive for research and development in the UAV field are the U.S. Department of Defense's (DoD) efforts to provide a more efficient and capable force for it's military forces. Currently, UAVs operating as remotely piloted vehicles (RPV) are utilized around the globe to provide intelligence, surveillance, and reconnaissance (ISR) as well as for small scale offensive actions. The immediate success of those operations has inspired the DoD to push further into the uncharted territory of complementing the

modern warfighter's emergent needs with UAV technology. The next step is to provide partially to fully autonomous UAV systems that have the ability to execute any peacetime or combat missions in support of desired "Effects Based Operations" (EBO). Such UAV operations not only have the potential to provide more fiscally attractive solutions to EBO needs, but since it offers the potential to remove the human from the most dangerous and dull aspects of the mission, UAVs offer the potential for dramatic improvement in organizational concepts, civilian or military.

The Air Force Institute of Technology's (AFIT) Advanced Navigation Technology (ANT) Center has recognized the importance of research in the autonomous UAV domain with ongoing projects in guidance and control of small aircraft (for definition of "small UAVs" see Roadmap, 2002:62). The ANT Center now has the foundation for autonomous UAV study including analytic research, MATLAB simulations, Hardware-In-The-Loop (HITL) Simulations, and flight test and demonstration. This broad capability, established through previous theses (Jodeh, 2006), allowed for the current research in this, and related theses. For this thesis, the primary tool utilized for the autonomous control research in the ANT Center was an off-the-shelf commercial autopilot provided by Cloud Cap Technologies, named the Piccolo II (Vaglienti, 2005).

In recent years, developing, simulating, and flight testing robust autonomous UAVs has been the topic of interest at numerous civilian universities/institutions throughout the country. However, when specifically dealing with small aircraft and autonomous control (esp. with the Piccolo II) there are only a few establishments conducting in-depth analysis, which includes the Autonomous Intelligent Networks and

Systems (AINS) Center for Collaborative Control of UAVs at the University of California, Berkeley (Girard, 2002 and Frew, 2004), the GRASP Laboratory at the University of Pennsylvania (Bayraktar, 2004), and the Aeronautics and Aerospace Department at the Massachusetts Institute of Technology (King, 2004 and Tin, 2004). These institutions have produced research which has advanced the control and manipulation of single and multiple UAV systems (King, 2004), dramatically pushing the envelope in this field. However, most of the previous research has, at best, glossed over the primary focus of this thesis; specifically, the affects of wind on the flight paths of the UAVs. The issue may have been mentioned, but prior research has not delved into the implementation of a robust system that continuously updates any wind correction parameters – a necessity for operational relevance.

The importance of this ability to strictly track a predetermined path becomes evident when dealing with current implementation of UAVs in the modern combat zone. Recent operations have shown the need for this technology to enable operations and navigation in the "urban canyon" environment. This demand requires tight adherence of point to point waypoint following. Moreover, urban buildings, streets, and the general environment generate unique and highly variable wind patterns which present a particular challenge for small, lightweight UAVs. The inherent strong up/down-drafts coupled with horizontal gusts can easily force a UAV off course and into an obstacle. Detailed studies on the topic can be found in (Cionco, 2004) and (Brown, 2003).

The research community generally characterizes the "wind effect" problem as an easily correctable issue through basic math. While it is true that the math involved was not drastically complicated, the difficulty lies in the implementation of these corrections

into the UAV autopilot systems – especially for the cost effective off-the-shelf systems. Most current systems will correct for a "static" wind reading, possibly at some ground station, and then employ this correction to the aircrafts control algorithm throughout the entire flight. However, as mentioned, in the new urban flight environment this methodology will not provide sufficient precision. Therefore, a continuously updating wind correction feeding the aircraft's control devices is not only desired, but required for the intricate demands of modern day operations.

1.2 -- Problem Statement

The ultimate goal of this research is to provide AFIT, the ANT Center, and the research sponsor, AFRL/VA, with a well-documented investigation into robust wind correction algorithms for small UAVs. To meet the operational needs, these schemes must continuously calculate the current wind corrections required and then update the UAVs flight plan to accommodate the local and constantly variable winds so as to assure the UAV remains on course or on target. The research platform supports UAVs flying in a constant or variable wind environment using Cloud Cap Technology's Piccolo II autopilot system. This problem statement has two primary parts. First, produce an adaptable algorithm for determining the current wind effects on the vehicle and the required heading and airspeed to compensate for that wind. Second, produce sensible approaches of implementing wind compensation algorithms on Commercial Off-the-Shelf (COTS), waypoint guided autopilots without hardware or software modifications to the autopilot or UAVs. In this thesis, the implementation will be demonstrated using a Piccolo II autopilot and the corresponding Software Development Kit (SDK).

Furthermore, simulated and actual flight test results were conducted to validate the algorithms.

1.3 -- Research Objectives

- Develop and document a wind velocity and direction determination scheme to be utilized on small UAVs in autonomous flight mode.
- Develop and document an interface algorithm in order to implement modifications to the flight path of the UAV to compensate for wind. The resulting ground track should show an improvement in the waypoint targeting and/or track following capabilities of the UAV.
- Demonstrate the performance of the algorithms through comparisons of unmodified and modified flight plans using HITL simulations as well as actual flight test data.

1.4 – Significance of Research

The significance of this research is to provide AFIT, the ANT Center, and AFRL/VA with a basis for continuing work in the precise navigation field of UAV technology. This research provides a robust manner in which to compensate for the common issue of variable winds. The current autopilot system incorporates wind finding calculations and adjustment techniques; however, the method used did not allow for a real time update of the wind. Therefore, the adjustments did not correct for dynamics of winds in the “urban canyon” or similar environments as efficiently as would be needed for combat operations.

Providing the foundations for a two dimensional, continuously updating wind correction algorithm allows for a starting point to delve into the more complex issues of precise, three dimensional track and waypoint control for lightweight, autonomous UAVs. This end goal is undoubtedly a few years in the future, but the reported research overcame the initial steps to improve the current systems.

The capability for the United States to, at will, deploy autonomous UAVs in an urban environment to conduct ISR or offensive operations will be indispensable to achieving the goals of EBO. To efficiently carry out a desired mission mitigating the risk of the loss of human life is the top level objective in this environment. The capacity to accurately infiltrate an unknown urban environment with a UAV will certainly contribute to those overarching objectives. This research will prove to be a significant step in that maturation.

Moreover, the concurrent AFIT studies of multiple, autonomous UAV formation flight (McCarthy, 2006) and UAV Autonomous Situational Awareness and Synthetic Vision (Dugan, 2006) provide further insights to enhance the goals of AFIT and the ANT Center.

1.5 – Methodology

The methodology varied for each of three research objectives. The calculations for determining the current wind conditions were developed through a manipulation of the difference in the GPS ground track and the actual aircraft magnetic heading. Utilizing basic trigonometry and algebra a wind direction and velocity were solved for, providing the current wind effects on the vehicle. Then, the new flight conditions, such as the

magnetometer heading and true airspeed (TAS), could be solved for. Additionally, these calculations were completed at continual time intervals; therefore, providing updating wind and correction estimates.

Once the wind-compensated values were known, there were three approaches for relaying that information back to the autopilot.

1. The more direct method of sending a new turn rate command coupled with the new TAS command. The difference between the actual and desired headings divided by a reasonable time step resulted in the turn rate command.
2. A second approach was to insert a new, updating waypoint which was placed at the correct heading to result in the overall aircraft ground track, after the effects of wind, to follow the original path to the original waypoint.
3. A unique approach to wind correction was employed by analyzing the ground footprint location of a nose mounted sensor. Despite precise navigation by the UAV, a sensor would not survey a target, but rather some undesired position offset from the target due to the difference in magnetic heading and the ground track direction. In order to correct this problem, the aircraft's flight path would be modified in order to counteract the sensor offset.

Developing the interface that implemented the wind correction algorithms on the Piccolo II autopilot involved using the Software Development Kit (SDK), provided by the manufacturer, to generate a C++ program. The SDK gave the operator real-time access to telemetry data from the autopilot. It also enables information to be sent back to the autopilot in order to update a desired parameter. Because this Software Development

Kit was provided by the same company as the autopilot, the interfacing occurs relatively smoothly whether this autopilot was in a HITL or in the airborne UAV.

The procedures for the HITL simulations and the actual flight testing were those formulated by Capt. Nidal Jodeh in his research from 2005-2006 (Jodeh, 2006). Essentially, the flight tests would first be run using the HITL simulator to ensure proper flying attributes. Then, the test team would fly the UAVs on Area B test range at Wright Patterson AFB, per the rules and regulations explained later.

With the algorithms effectively manipulating the flight path, the modified path results were compared to the original results using a MATLAB script developed previously (Jodeh, 2006) and then adapted by the author. This program output two dimensional (also 3-D, if desired) plots of the aircraft's true flight path, simulated or actual, in relation to the desired waypoints and flight paths. From these figures, the variations were easily analyzed.

1.6 – Thesis Preview

Chapter II details the equipment utilized including the aircraft components, the avionics components, the autopilot, and the simulation components and provides a background on the flight testing, as a whole. Chapter III methodically looks at the equation build ups and the varying attempts at the implementation of the modified flight parameters. Chapter IV presents the results of the baseline tests, the HITL simulations, as well as the actual flight tests. Chapter V summarizes the conclusions and recommendations.

II. Background

2.1 – Overview

Chapter II provides background information on the specific equipment, components, and the flight testing procedures utilized in the formulation of the wind compensation algorithms. Thus, it supplies the reader the necessary information to understand the remaining chapters. Initially, the airframe, engine, and propeller are discussed. This is followed by a discussion of the avionics systems, including the standard radio controller (RC), the autopilot, and the digital magnetometer. Next, the Hardware-In-The-Loop (HITL) simulation setup is detailed along with the Software Development Kit (SDK) interface. The chapter concludes with a description of the flight testing setup, procedures, and the data telemetry collection and handling.

2.2 – Aircraft

2.2.1 – Airframe

The aircraft used for this research was the ANT Center's Rascal 110 R/C aircraft constructed by the SIG Manufacturing Company, Inc. This aircraft provided a rugged platform with a relatively abundant amount of interior volume, stable flight characteristics, and simple construction techniques. The Rascal 110 is a high wing, "tail dragger" configuration that was delivered in an Almost-Ready-to-Fly (ARF) configuration. Prior to delivery SIG constructed most of the fuselage and wing structures out of thin plywood, balsa wood, aluminum, and fiberglass. The ANT Center then completed final assembly of the components and modified the interior as needed. A key modification was the addition of a 50 oz fuel tank, to provide a flight time of approximately two hours. About 40 hours of work was required to complete the aircraft

in the desired configuration. Figure 1 shows a completed version of the ANT Center's Sig Rascal 110's.

Figure 1. Two Completed Sig Rascal 110's (Jodeh, 2006)

The manufacturer provided airfoil was a combination of two Eppler planforms. The top airfoil surface is an Eppler 193, while the bottom is an Eppler 205, joined at the chord lines. SIG also stated that the resultant section thickness was 11.5% of the root chord with an aspect ratio of 6.875:1. However, through previous research, Air Force Captain Nidal Jodeh found the aspect ratio to be 7.94 when assuming a semi-elliptical planform as opposed to the rectangular assumption used by the manufacturer (Jodeh, 2006). Unfortunately, SIG Inc. did not provide any stability, performance, weight, balance, or aerodynamic data with the Rascal 110. Capt. Jodeh determined most of those values through his research (Jodeh, 2006). Figure 2 displays the wing planform view of the Rascals.

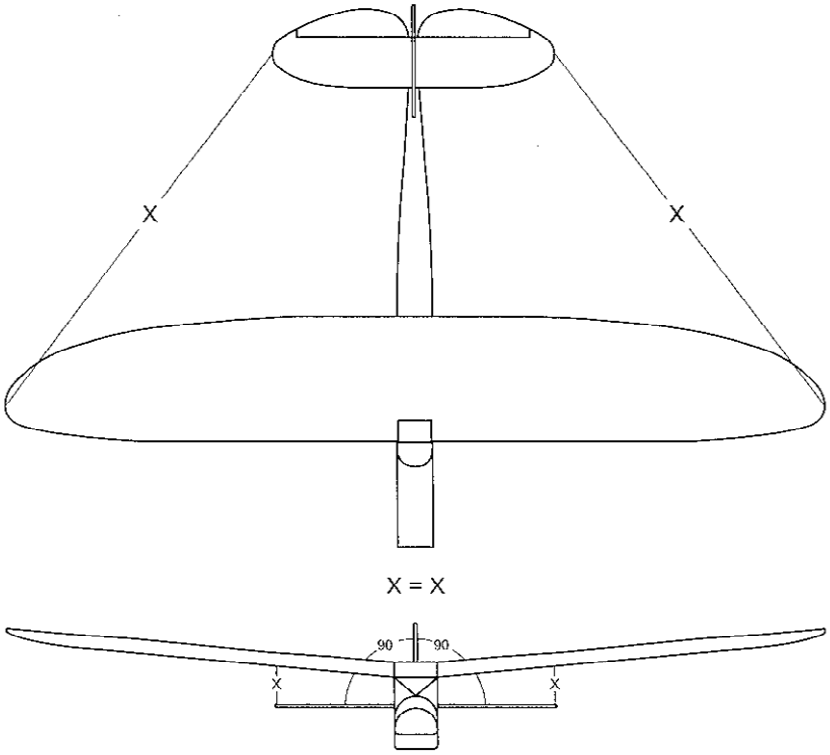

Figure 2. Sig Rascal Wing Planform View (Jodeh, 2006)

Table 1, below, outlines the pertinent aircraft data and parameters that characterize the Sig Rascal 110.

Table 1. Various Sig Rascal 110 Characteristics

SIG RASCAL PARAMETER	VALUE
Wing Span	9.16 ft
Aspect Ratio	7.94 ft
Aircraft Mass (Empty Fuel Tank, Engine, Reciever)	14.19 lbf
Gross Takeoff Weight (GTOW)	18.74 lbf
Length (including Engine & Tails)	76 in
Payload	~10 lbf
Normal Operating Airspeeds	12-30 m/s (true)

2.2.2 – Engine and Propeller

The SIG Rascal 110s used by the ANT Center are powered by FS-120S III four cycle engine produced by O.S. Engines. The power plant came ready to use, including a diaphragm fuel pump, matching carburetor, and a built in pressure regulator. The 1.218 cubic inch engine's output was rated at 2.1 brake horsepower (bhp) at 12,000 revolutions per minute (rpm). To translate the horsepower to thrust, the engine was combined with a 16x8 synthetic propeller from the APC Company. This 32.5 ounce power plant was capable of pulling the Rascal at over 60 knots. Figure 3 and Figure 4 display the O.S. engine and the APC propeller (O.S., 2003 and APC, 2006).

Figure 3. O.S. FS-120S III Four Cycle Engine

Figure 4. APC 16x8 Nylon Propeller

2.3 – Avionics

The avionics utilized by the ANT Center in the Rascal 110's had three separate components, the basic radio control (RC) system, the Piccolo II Autopilot System, and the digital magnetometer.

2.3.1 – Radio Control System

The RC system was a Futaba 9CAP/9CAF 8 channel transmitter coupled with a Futaba R149DP PCM 1024 receiver. High torque servos, also Futaba products, translated the radio signals to movement in the control surfaces. Figure 5 is a photo of the advanced Futaba transmitter (Futaba, 2006).

Figure 5. Futaba 9CAP/9CAF 8 Channel Transmitter

2.3.2 – Piccolo II Autopilot

The Piccolo II autopilot system, which was the crux of this research project, was purchased from Cloud Cap Technologies. This unit is well suited for incorporation into small UAVs, providing a completely autonomous aircraft capable of navigating through a flight plan of predefined or real time updated waypoints. The entire setup included the autopilot, the ground station interface, the manual control box, the HITL components, and software.

The autopilot box provided attitude data through three gyros and two double-axis accelerometers for rate and acceleration measurements of the aircraft. The autopilot uses a Kalman filter to estimate attitude and gyro bias using a GPS-derived pseudo-attitude as the measurement correction (Vaglienti et al. 2005). The pitot-based flight data, true airspeed (TAS), absolute altitude, and outside air temperature (OAT), were delivered via a dual ported 4kPa dynamic pressure sensor, and an absolute ported barometric pressure sensor, and a board temperature sensor (Vaglienti et al. 2005). The Piccolo II autopilot utilized a 40 MHz Motorolla MPC555 PowerPC for all processing (Vaglienti, et al. 2005). Position data was provided through an imbedded GPS unit. The wireless link used to transfer the command and control, telemetry, payload, differential GPS corrections, and pilot in the loop information was a 1W 900MHz and 1W 2.4GHz radio modem at up to 40 Kbaud of throughput (Vaglienti et al, 2005). The GPS receiver was a 16 channel receiver with 8192 simultaneous time-frequency search bins and a 4 Hz position update rate (u-Blox, 2005). The physical, on-board unit was 2 inches wide by 2.5 inches high and 5.25 inches deep, totaling 26.25 inches^3 in volume. The box was constructed of electromagnetically shielded carbon fiber. Figure 6 illustrates the block

diagram of the complete avionics suite inside the Piccolo II system. Figure 7 is a picture of the Piccolo II on-board autopilot (Vaglienti, 2005).

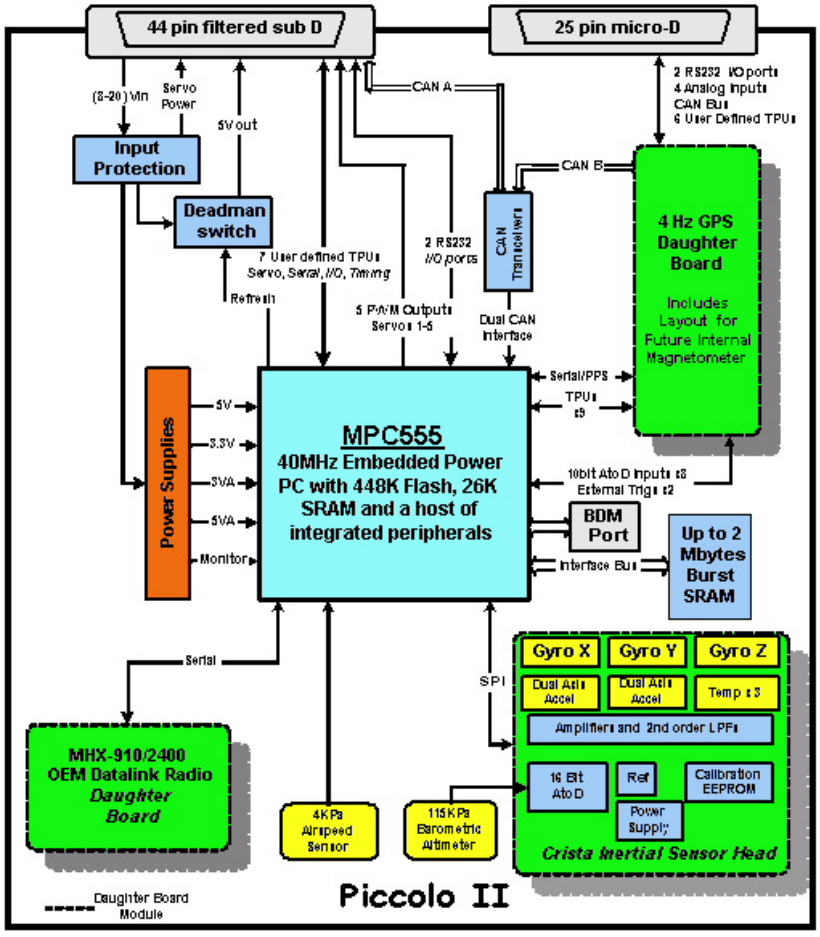

Figure 6. Piccolo II Block Diagram of Internal Components

Figure 7. Piccolo II Airborne Autopilot Unit

The ground-based equipment required to interface and control the airborne unit include the Ground Station interface, a laptop computer, RC control box, and the UHF and GPS antennas. The Ground Station software interface, known as the Operator Interface, ran on a laptop PC and was the primary command and control device. The aircraft telemetry, GPS tracking, component statuses, and control surface gains were all available through the Operator Interface. The RC control box ensured the pilot's ability to take control of the aircraft at all times. Essentially, it provided a direct pilot-in-the-loop interface using the Piccolo II autopilot as the RC receiver. Detailed procedures and instructions on the effective use of the Operator Interface was written and provided (online) as the Piccolo System User's Guide Version 1.3.0 from Cloud Cap Technology, written by Vaglienti et al. (2005). The RC box and the remaining electrical components required for this system were all collocated in the Ground Station. Figure 8 presents the entire arrangement of the required ground equipment for the Piccolo II system (Vaglienti, 2005).

Figure 8. Required Ground Equipment (minus the laptops) for the Piccolo II Autopilot System

An important component used in the implementation of the Piccolo II autopilot was the Fail Safe Control Relay. This enabled the UAV pilot to simply toggle between standard RC control and the Piccolo's manual/autonomous control. Additionally, the Fail Safe Control Relay switched from the autonomous mode to RC mode, and vice-versa, if the control signal strengths dropped below predetermined levels. As an example, if the UAV was under autopilot control and the signal was lost, for any reason, the relay was activated and RC control was implemented (also, if under RC control and RC signals are lost, autonomous mode would be engaged). The designers of the fail safe, William J. Schmoll and Richard Marker of Air Force Researh Labs Sensors Directorate (AFRL/SN), detail the system in the following:

> "The channel 8 output of receiver A goes to the monostable multivibrator 74C221 trigger. The 15k ohm resistor, the 5k ohm potentiometer, and the 0.2 uF capacitor

form the external timing circuitry for the 74C221. The multivibrator is adjusted by the 5k ohm potentiometer for exactly 1.5 milliseconds. The channel 8 pulse al goes to the 74C175 flip-flop's "D" input. When the monostable pulse ends (goes low) the output of the 74C175 is latched in the state of the channel 8 pulse. If the channel 8 pulse is longer than 1.5 msec then the 74C175 output will be high and if shorter than 1.5 msec then it will be low. The output of the 74C175 goes to the select inputs (pin 1) of the 74C157 data selector chips. If "Select A/B" is low, receiver A (R/C) is selected and if high the receiver B (autopilot) is selected." (Jodeh, 2006)

Figure 9 is a schematic of the Fail Safe Control Relay (Jodeh, 2006).

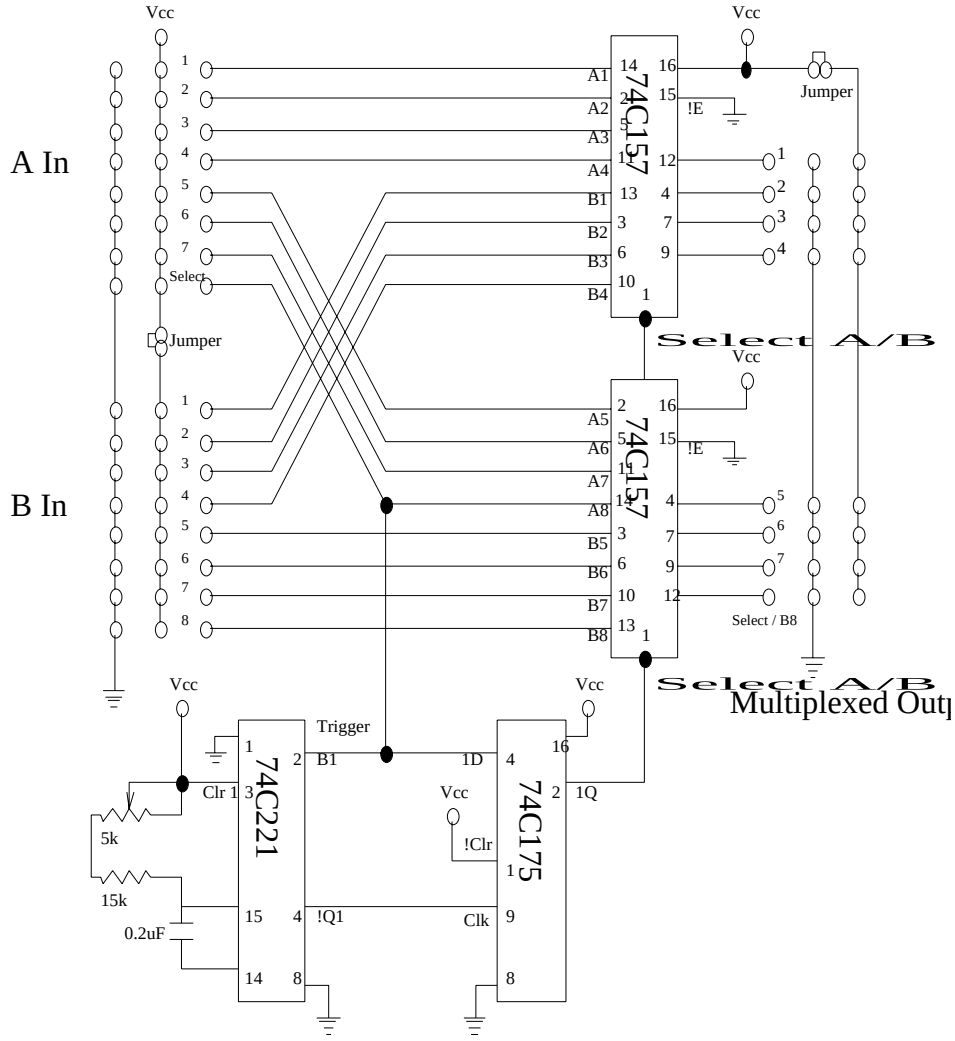

Figure 9. Fail Safe Control Relay Schematic

Figure 10 is the block diagram depicting the air and ground avionics and communication paths (Jodeh, 2006).

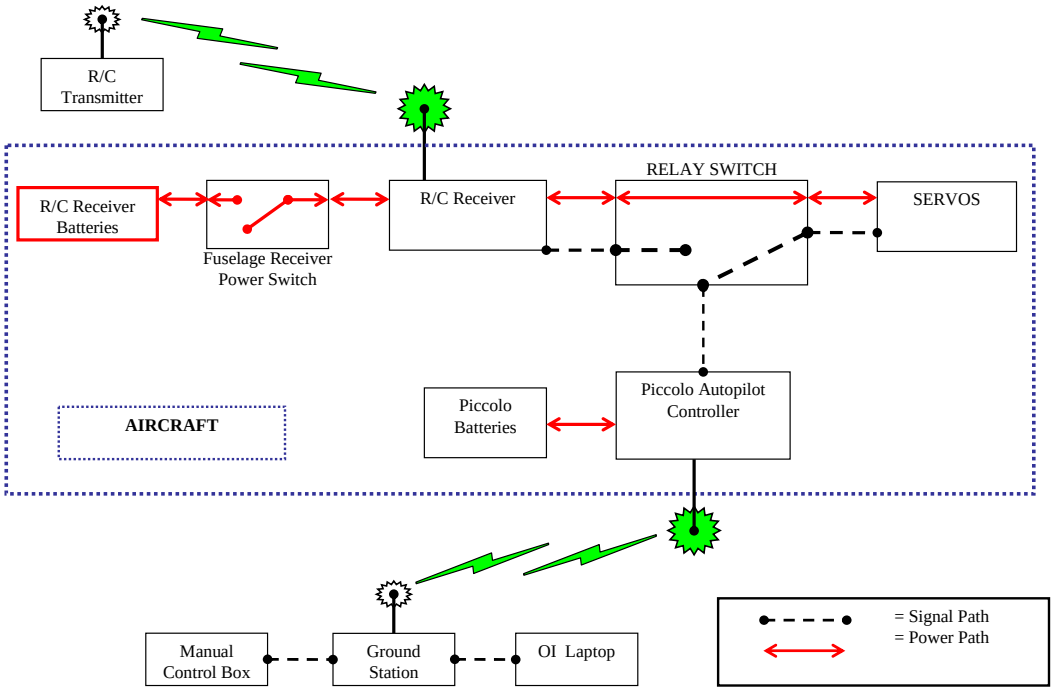

Figure 10. Complete Autonomous Flight Setup

2.3.3 – Honeywell HMR2300 Digital Magnetometer

The key component added for this research was the Honeywell HMR2300 Smart Digital Magnetometer. Whether simulated or actual, this device allowed the team to observe the magnetic heading of the aircraft. This was essential in determining the UAV's crab angle, which made it possible to continuously estimate the winds. The GPS telemetry provided the ground track direction, while the magnetometer provided the true heading of the aircraft – the difference being that crab angle. Measuring 4.2 x 1.5 x 0.876 inches, the Honeywell unit was easily mounted in line with the Rascal's nose in the

forward portion of the internal equipment bay. Because Cloud Cap Technologies recommended this specific unit, clear directions were provided in the Piccolo User Manual to calibrate and integrate the magnetometer with the Operator Interface. Figure 11 is a photograph of the device as provided on the Honeywell website.

Figure 11. Honeywell HMR2300 Digital Magnetometer (Honeywell, 2004)

2.4 – Simulation

The primary means of preliminary evaluation for any flight testing is through a complete system level simulation in which the highest fidelity model is desired, if not required, to produce accurate results. From the simulation data, the researchers can then make reasonable assumptions on how the test object will behave under real world conditions. For this project, the proven method of Hardware-in-the-Loop (HITL)

simulation was utilized. Here, the actual device, the Piccolo II autopilot, was placed directly in the simulation loop. Then, the autopilot interacted with the simulated aircraft (produced on the provided Piccolo Simulator) as if airborne.

2.4.1 – Hardware in the Loop (HITL)

As mentioned above, the HITL simulation involved the interaction of multiple simulated and/or real components, including the Piccolo Aircraft Simulator, the Piccolo II Autopilot, the Ground Station box, and the Operator Interface. (As a note, due to the system operational requirements, two desktop and/or laptop computers were employed.) Figure 12, below, presents a graphical representation of the Hardware in the Loop Simulation setup in the ANT Center (Jodeh, 2006).

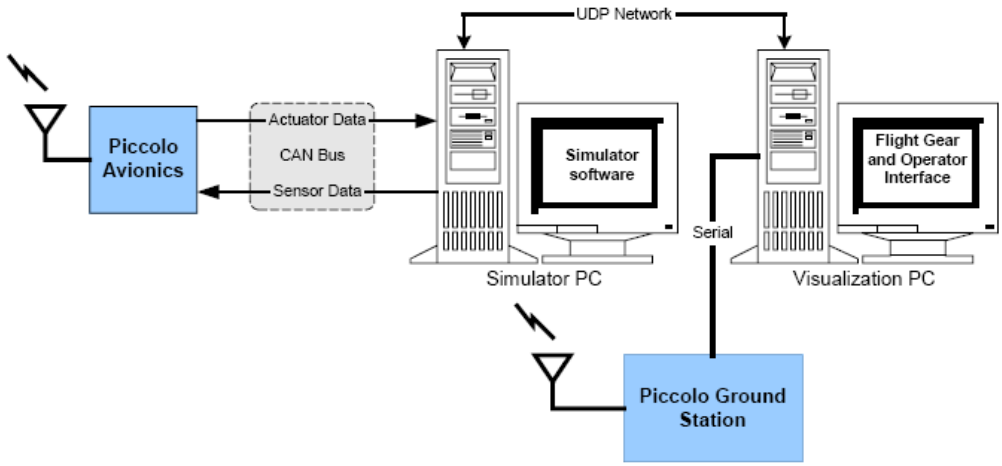

Figure 12. Standard Hardware in the Loop Simulation Setup

The two computers designated for the HITL simulations in the ANT Center were COTS and of average computing power. One of the HITL computers was used to run the Operator Interface while the other was used to run the aircraft simulation. The Operator Interface allowed the autopilot settings to be viewed and/or altered, as well as presenting

a bird's eye view of the aircraft, simulated or actual, and its progression along the flight plan track. The Ground Station box was connected to a serial port on the computer running the Operator Interface. This connection provides the user with an interface to the ground station so signals and telemetry could be relayed to the autopilot over a wireless transmission. The GPS and UHF antennas were plugged into the Ground Station Box. Next, the Piccolo II was connected to the computer running the aircraft simulation provided by Cloud Cap through its main harness. The simulation then had the ability to send the simulated aircraft sensor data to the autopilot unit so as to replicate actual aircraft motion.

Additionally, the recommended (by Cloud Cap) flight visualization software package, "Flight Gear," was occasionally run on the Operator Interface computer as well. This program enabled increased situational awareness compared to the top-down view provided by the Operator Interface. Flight Gear provides three dimensional top, trail, pilot, or wingman views. Yet, the purpose of the this research was to analyze, and then better, the 2-Dimensional, cross-track wind flying capabilities of the UAVs; thus, for most situations, the top view sufficed and the Flight Gear software was not employed.

2.4.2 – Software Development Kit (SDK)

Cloud Cap Technologies recognized that modifications to the Piccolo II was an idea that many of its autopilot users might desire. Thus, they provided a Software Development Kit (SDK), in the form of C++ code, to facilitate such modifications. During the summer of 2005, AFIT employed Randall Plate, a local college student, as an intern in the ANT Center. His primary goal was to experiment with the Piccolo SDK.

This work provided important insight regarding as how to efficiently perform modifications to the Piccolo C++ code and the resultant effect on the autopilot. By the end of his term, Mr. Platte was able to provide C++ code, with comments, that allowed the user to interface with the autopilot in real time. Although the code was preliminary, it established a foundation to build upon for many the current ANT Center UAV projects – this one included.

As the Piccolo II operates, it actively creates and logs packets of information that are transmitted to and from the ground station. The Software Development Kit enabled the user to essentially intercept, modify, and then send back modified data packets. In summary, this was how modifications were applied to an operational autopilot unit. In this case, an initial function was coded to continuously estimated the wind as the aircraft flew. Next, a series of functions implemented the desired corrections based on those estimated wind velocities and directions. Finally, a group of functions were used to remit the data back to the Piccolo II. The effects of those modifications were viewed, in real time, through the Operator Interface.

2.5 – Flight Testing

2.5.1 – Overview of Flight Test

The flight testing of any aircraft is an absolute necessity to ensure that the behavior and performance are within predetermined specifications regardless of whether the system is totally new or simply modified. This project was no exception, and served

flight tests were conducted to validate the wind finding and correcting techniques as applied to the proven SIG Rascal 110 outfitted with the Piccolo II autopilot.

A myriad of organizations have flown and proven the stability and performance of the SIG Rascal, the autopilot, and the combination of the two. The ANT Center completed this first step through Capt. Jodeh's thesis research on the development of autonomous UAV system (Jodeh, 2006). This allowed for only a cursory check flight of the aircraft which included basic airworthiness checks by means of "standard maneuvers" in RC mode followed by a set of autonomous tracking maneuvers. With the enabling parameters performing as expected, the test conductor and the UAV pilot began the designated flight tests for that session. Upon completion of the experiment, the test conductor stopped the Operator Interface program and captured the logged telemetry. Once back in the lab, that set of data was processed and analyzed. Chapter V details the specific flight tests and their objectives.

2.5.2 – Flight Test Range

Consistent with standard protocol for the testing of official government property, this research testing was planned for and conducted on government land. All test flights were planned to be flown on Area B of Wright Patterson Air Force Base (WPAFB) in Dayton, Ohio, specifically, on and around the closed runway 27, located in the southwest corner of Area B. This area is approximately 1.5 miles in length and one mile wide, with a 400 foot above ground level (AGL) ceiling. The field elevation was 785 feet mean sea

level (MSL), making the ceiling for flight tests 1185 feet above MSL. This area was also occupied by other facilities conducting autonomous UAV flight tests.

Figure 13 is an aerial view of the Area B test site. The approximate boundaries of the test area are outlined by the heavy, dashed-line trapezoid (Jodeh, 2006).

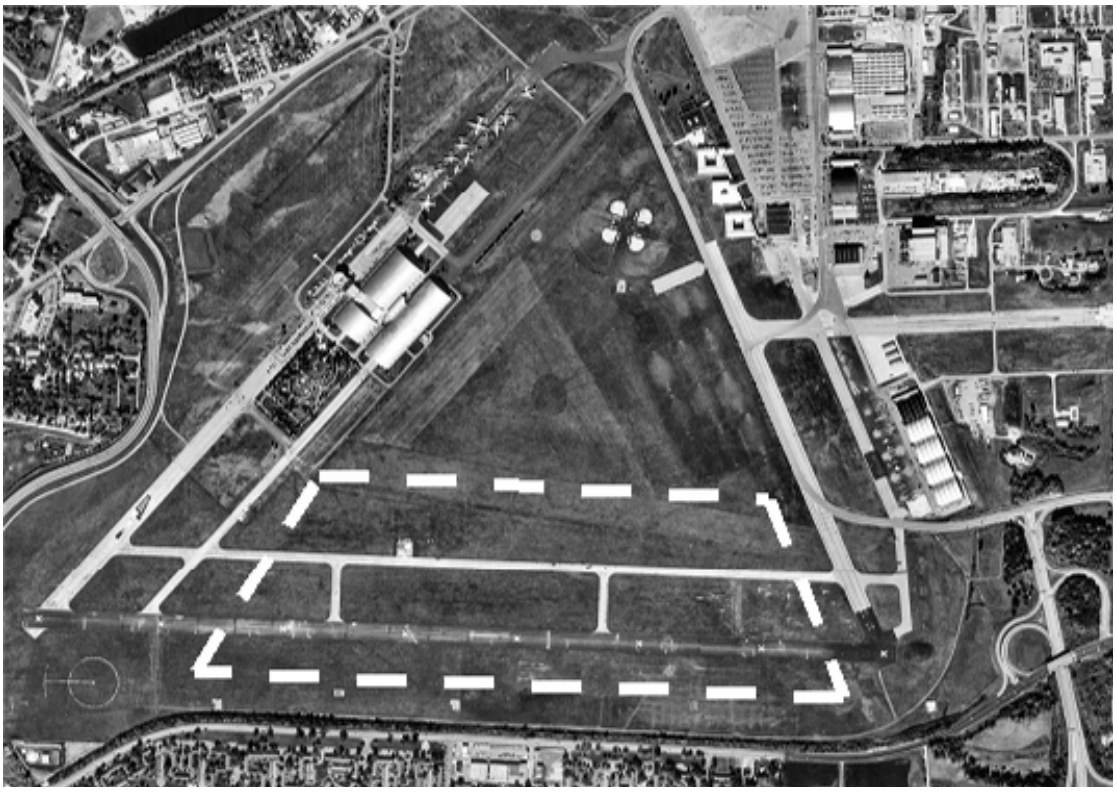

Figure 13. WPAFB, Area B Flight Test Range

2.5.3 – Ground Equipment

The test team's ground equipment was consolidated in a 20- foot trailer, which then took on the role of a test operations center. An external, gasoline powered generator provided the AC electricity to power the computers, the Ground Station box, the battery charging equipment, etc. The UHF and GPS antennae were attached to trailer's roof as was an orange windsock. Additional equipment, including folding chairs and tables, small tool kit, two-way radio headsets, packed comfortably into the trailer. Similarly, miscellaneous equipment including an RF meter, cones, fire extinguisher, spill kit, first aid kit, video camera, battery testers, and a handheld GPS unit were staged and stored in the trailer. Moreover, a 10-12 foot desk was mounted on the interior to facilitate workstations for the Ground Station, computers, etc. As opposed to the desktop computers utilized in the ANT Center's HITL simulations, the "field" setup for flight test exploited laptop computers. Figure 14 shows the open rear of the test trailer and the normal test team which was comprised of four to five members, including the pilot (contracted from Wyle Laboratories), the test conductor, and spotters/observers (Jodeh, 2006).

Figure 14. Ground Equipment and Test Team Conducting a Flight Test

2.5.4 -- Criteria for Flight Test of UAVs at WPAFB

Due to proximity of the test range on Area B to other facilities, government and civilian, certain flight test restrictions and safety of flight criteria were imposed. The Configuration Control Board (CCB), Technical Review Board (TRB), and Safety Review Board (SRB) were administered by AFIT and AFRL personnel, per the Air Force base regulations to ensure safe operation within controlled airspace. Table 2 lists the prominent criteria for flight testing in the Area B range.

Table 2. Prominent Criteria for Flight Tests (Jodeh 2006)

Winds Less than 30 mph
Temperature Greater than 40^{o} F
Visibility Greater than 3 Miles
Cloud Ceiling Minimum 500 ft AGL
Airspace Ceiling Maximum 400 ft AGL
GPS Satellites 6 or more visible
Radio Frequency Interference Check
Safety Equipment and First Aid Kit
Pitch, Roll, and Yaw Rate Gyro Operations
Static and Dynamic Pressure Port Operation
WPAFB Control Tower Notification

2.5.5 – Wind Correction Implementation

Consistent with standard flight test protocol, the wind correction flight tests conducted were planned in an order that gradually increased test complexity and challenge. Similarly, testing was begun on a mildly breezy day and worked up to a day when the winds were 35%-50% of the aircrafts velocity. This limit was deemed suitable since it is generally accepted that small UAVs would not be able to effectively operate in an environment with sustained winds of greater than 50% of its normal cruising speed.

2.5.6 – Data Collection and Handling

At the conclusion of a flight test, the Piccolo's telemetry was logged, in ASCII format, in the Operator Interface folder on the respective laptop. The software acquired and stored 70 parameters that were continuously updated at a selected data rate. The two data rates available were "Request Slow" at 1 Hz and "Request Fast" at 20 Hz. The rate

chosen by the test conductor was determined by the fidelity required. Additionally, the individual telemetry files were only created when the Operator Interface was turned off.

Two methods were used to transform the flight data to usable plots and values. First, the telemetry file was opened in Microsoft Excel, placing each of the 70 parameters in its own column. At this point, the analyst would delete any unnecessary rows and columns in order to reduce the file size. For example, a half an hour flight test at the "Request Fast" rate would produce an Excel file with approximately 60,000 rows by 70 columns, or 4,200,000 data cells. Trimming the excess parameters could reduce the number of data cells by as much as two-thirds. The modified Excel file was imported into MATLAB and saved as a MATLAB ".mat" file. This new file was then uploaded into a program which displayed two- and three-dimensional plots of the aircraft's actual track in relation to the desired. Additional plots to show various flight measurements and wind values, created by the author, supplemented this program. The program is attached in the Appendix C.

A second method of data acquisition was developed during the course of this research. The SDK was manipulated such that it would output only the desired telemetry in a Microsoft Notepad file. Then, similar to above, this file could be imported to either Excel or directly into MATLAB to be exploited by the same plotting program discussed above. Table 3 lists the 70 parameters available through the SDK.

Table 3. Available Telemetry through the Piccolo SDK

1. Clock [ms]	25. Static [Pa]	49. Surface7 [rad]
2. Year	26. Dynamic [Pa]	50. Surface8 [rad]
3. Month	27. P [rad/s]	51. Surface9 [rad]
4. Day	28. Q [rad/s]	52. P_Bias [rad/s]
5. Hours	29. R [rad/s]	53. Q_Bias [rad/s]
6. Minutes	30. Xaccel [m/s/s]	54. R_Bias [rad/s]
7. Seconds	31. Yaccel [m/s/s]	55. AP_Global
8. Latitude [rad]	32. Zaccel [m/s/s]	56. PDyn_Stat
9. Longitude [rad]	33. Roll [rad]	57. Alt_Stat
10. Height [m]	34. Pitch [rad]	58. Turn_Stat
11. Ground Speed [m/s]	35. Yaw [rad]	59. Flap_Stat
12. Direction [rad]	36. LeftRPM	60. Track_Stat
13. Status	37. RightRPM	61. PDyn_Cmd [Pa]
14. InputV [V]	38. WindSouth [m/s]	62. Alt_Cmd [m]
15. InputC [A]	39. WindWest [m/s]	63. Turn_Cmd [rad/s]
16. FirstStageV [V]	40. WindError [m/s]	64. Flap_Cmd [rad]
17. FiveDV [V]	41. RSSI	65. Track_Cmd
18. FiveAV [V]	42. Surface0 [rad]	66. MagHdg [rad]
19. CPUV [V]	43. Surface1 [rad]	67. SonicAlt [m]
20. GPSV [V]	44. Surface2 [rad]	68. AckRatio [%%]
21. BoxTemp [C]	45. Surface3 [rad]	69. ServoV [V]
22. Altitude [m]	46. Surface4 [rad]	70. ServoC [A]
23. TAS [m/s]	47. Surface5 [rad]	
24. OAT [C]	48. Surface6 [rad]	

2.6 – Chapter Summary

This chapter provided a review of the equipment utilized and the overarching techniques applied to conduct this research program. The SIG Rascal 110 powered with the O.S. FS120S-III carried the Piccolo II autopilot. The avionics package included a sophisticated 8 channel transmitter and receiver produced by Futaba, the autopilot components, and the fail safe relay. The flight tests were conducted on Area B of Wright

Patterson AFB in Dayton, Ohio and adhered to all of the rules and regulations outlined. Additionally, flight data was analyzed using Microsoft software coupled with MATLAB.

III. Development of the Wind Correction Approaches

3.1 – Overview

The overall impact on flight path trajectory effects due to wind on small UAVs were best viewed from overhead. This perspective allowed for ground tracks, airborne magnetic headings, correction angles, and relative distances to be determined using basic trigonometry. The bulk of mathematics behind this research utilized manipulations of sin/cosine theory, Pythagorean Theorem, and basic Dynamic Inversion.

3.2 – Real Time Wind Estimating

The first step was to determine the wind heading and velocity so the aircraft's heading, velocity, flight path, etc. could be adjusted to compensate for the wind. The Piccolo II autopilot allowed the operators to not only view, but capture (via the SDK) many of the variables required in this compensation. However, one limitation of the Operator Interface was that the physical display only showed the resulting ground track of the aircraft. The difference in the aircraft magnetic heading and the resulting ground track produced an angle, known as the "crab" angle. Thus a separate scheme was required to determine the crab angle.

The basic Piccolo II autopilot only displayed wind estimates at intermittent updates or when designated "Wind Interval Turns" were commanded. In real world applications, it is rare for the winds aloft to be constant, especially so in an urban canyon environment. Therefore, the need for a real time, updating wind estimate became abundantly clear. Fortunately, Cloud Cap recognized issues such as this and provided their SDK to allow modifications or additions to the autopilot's functions. Thus, the

following equation methodology was implemented in the SDK using C++ programming to provide a real time wind estimate.

Using a vector component break down, three aspects to the flight path of the UAVs were identified. The aircraft itself had two velocity vectors: one based solely on the airborne vehicle's orientation and the other being the ground track. Each of these had velocity magnitude and angle components. The presence of wind was then characterized as the difference between the two aircraft velocity vectors. Equations 1 and 2 show that the aircraft's heading (θ_{MAG}) and true airspeed (V_{TAS}) plus the wind effects (V_W and θ_W) will result in the overall ground track (V_G and θ_G). Note, all angles, θ, were measured clockwise from North = 0°.

$$V_{TAS}\sin(\theta_{MAG}) + V_W\sin(\theta_W) = V_G\sin(\theta_G) \quad (1)$$
$$V_{TAS}\cos(\theta_{MAG}) + V_W\cos(\theta_W) = V_G\cos(\theta_G) \quad (2)$$

Grouping all of the aircraft components on one side of these equations resulted in Equations 3 and 4. These were used as the base equations to begin the manipulations for solving the real time wind velocity and heading.

$$V_W\sin(\theta_W) = V_G\sin(\theta_G) - V_{TAS}\sin(\theta_{MAG}) \quad (3)$$
$$V_W\cos(\theta_W) = V_G\cos(\theta_G) - V_{TAS}\cos(\theta_{MAG}) \quad (4)$$

To simplify the equations, the substitutions shown in Equations 5, 6, 7, and 8 were made.

$$x = V_W \quad (5)$$
$$y = \cos(\theta_W) \quad (6)$$

$$a = V_G\cos(\theta_G) - V_{TAS}\cos(\theta_{MAG}) \quad (7)$$
$$b = V_G\sin(\theta_G) - V_{TAS}\sin(\theta_{MAG}) \quad (8)$$

Inserting the new variables, Equations 3 and 4 reduce to Equations 9 and 10.

$$a = (x)(y) \tag{9}$$

$$b = x\sqrt{1-y^2} \tag{10}$$

The next step was to simultaneously solve for "x" and "y." These two equations with two unknowns were easily solved using software such as MATLAB or by hand using classical mathematics. Equations 11 and 12 are the results.

$$x = \sqrt{a^2 + b^2} \tag{11}$$

$$y = \sqrt{\frac{a^2}{a^2 + b^2}} \tag{12}$$

Finally, the original wind variables were reinserted, solving for the wind velocity and wind heading.

$$V_W = \sqrt{a^2 + b^2} \tag{13}$$

$$\theta_W = \cos^{-1}\left(\sqrt{\frac{a^2}{a^2 + b^2}}\right) \tag{14}$$

3.3 – Turn Rate Approach Equations

Now that the wind variables were known the correction that needed to be applied to the aircraft to adjust for the wind could be deduced. As will be shown, there is more than one approach to implementing these corrections.

The most direct method utilized the mathematical principle of "Dynamic Inversion" to solve for a new aircraft velocity, V_{TAS2}, and heading, θ_{MAG2}, which could then be commanded through the Piccolo II to compensate for the wind. The dynamic

inversion principle essentially backs out a desired command based on of a known output variable. In this case, the output variables, V_W and θ_W, were solved for using the known input quantities, which were extracted from the Piccolo's telemetry. The desired ground track, a known value, and the wind variables, known parameters, were combined to back out the new inputs. Essentially, the end result is that the ground track was known and the corresponding inputs which would provide that desired output were then reverse engineered. The following procedure outlines this process.

Once again, Equations 1 and 2 were the baseline from which to start the calculations. However, this time, the winds are known based on the previous section and the aircraft's true airspeed and magnetic heading values required (to be commanded) to counteract the wind need to be solved for. These new values were denoted with an underscore "2."

$$V_{TAS2}\sin(\theta_{MAG2}) + V_W\sin(\theta_W) = V_G\sin(\theta_G) \quad (1)$$
$$V_{TAS2}\cos(\theta_{MAG2}) + V_W\cos(\theta_W) = V_G\cos(\theta_G) \quad (2)$$

The values being solved for were then isolated, resulting in Equations 15 and 16.

$$V_{TAS2}\sin(\theta_{MAG2}) = V_G\sin(\theta_G) - V_W\sin(\theta_W) \quad (15)$$
$$V_{TAS2}\cos(\theta_{MAG2}) = V_G\cos(\theta_G) - V_W\cos(\theta_W) \quad (16)$$

As in the case of the real time wind estimating, a similar change of variables was done to simplify the terms.

$$x_2 = V_{TAS2} \quad (17)$$
$$y_2 = \cos(\theta_{MAG2}) \quad (18)$$

$$d = V_G\sin(\theta_G) - V_W\sin(\theta_W) \quad (19)$$
$$c = V_G\cos(\theta_G) - V_W\cos(\theta_W) \quad (20)$$

The reduced equations were represented by Equations 21 and 22, below.

$$c = (x_2)(y_2) \tag{21}$$

$$d = x_2\sqrt{1 - y^2} \tag{22}$$

The solutions for the non-linear, simultaneous equations above were determined using MATLAB and hand calculations, just as before.

$$x_2 = \sqrt{c^2 + d^2} \tag{23}$$

$$y_2 = \sqrt{\frac{c^2}{c^2 + d^2}} \tag{24}$$

Replacing x_2 and y_2 with the original variables, the new true airspeed and magnetic heading were solved using Equations 25 and 26. This gives expressions for the true airspeed and magnetic heading as a function of the measured winds and desired ground track. Thus, commanding the UAV to fly V_{TAS2} and θ_{MAG2} will produce the desired ground track.

$$V_{TAS2} = \sqrt{c^2 + d^2} \tag{25}$$

$$\theta_{MAG2} = a\cos(y_2) \tag{26}$$

Ideally, this approach of solving for the new aircraft heading and airspeed would provide the most direct manner in which to implement new aircraft control commands. Initially, it seemed straightforward to continuously input these two new values to the Piccolo II, creating an updating correction. The new heading would be input as a turn rate, hence the name "Turn Rate Approach," and the airspeed would be commanded as a dynamic pressure. However, as will be detailed in section 3.6, the implementation of a

new airspeed and magnetic heading through the SDK created barriers that were beyond the scope of this thesis.

3.4 – Updating "Rabbit" Waypoint Approach

The second approach to wind effects correction was referred to as Updating "Rabbit" Waypoint Insertion. The methodology took the real time wind values determined above and attempted to insert a new, updating waypoint that would be offset from the original. The aircraft would then be commanded to fly to the adjusted waypoint; however, due to the wind drift it would never reach that point and instead end up at the original, targeted waypoint. The process below provides the framework for the "Rabbit" waypoint placement approach.

To begin with, the relative, horizontal distance, in meters, between the aircraft's current position and the current waypoint was required. The waypoints, as well as the aircraft's position, were provided in Latitude/Longitude/Altitude (LLA) format. Therefore, both positions were first converted to East/North/Up (ENU) coordinates using the preexisting code in the SDK. So, if D was defined as the straight line, ENU distance between the aircraft's location and current waypoint. Then inserting Equations 27 and 28 into the Pythagorean Theorem, the horizontal distance was determined and presented as Equation 29.

$$A = ENU_{East-A/C} - ENU_{East-Wypt} \tag{27}$$

$$B = ENU_{North-A/C} - ENU_{North-Wypt} \tag{28}$$

$$D = \sqrt{A^2 + B^2} \tag{29}$$

The overarching goal, or perhaps better stated as the "anti-goal," of the "rabbit" was for the UAV to continually chase the rabbit, but never actually catch it. To implement this aspect the new waypoint was repeatedly placed at a distance greater than "D." Next, the bearing, or angle, (from the aircraft) of the new waypoint had to be determined. This angle would not only depend on the real time wind velocity and direction, but also in which Cartesian quadrant the aircraft was located with respect to the original waypoint. The following set of equations progress through the operations required to not only find the correct angle and distance of the "Rabbit," but also place it using the correct ENU coordinates.

If (θ_G > 0 && $\theta_G \leq 90$);

angle_deg = θ_G - 90; (30)

abscos = $\left|D\cos(\text{angle\_deg})\right|$ (31)

abssin = $\left|D\sin(\text{angle\_deg})\right|$ (32)

$ENU_{East-NewWypt}$ = $ENU_{East-A/C}$ + abscos; (33)

$ENU_{North-NewWypt}$ = $ENU_{North-A/C}$ + abssin; (34)

If ($\theta_G > 90$ && $\theta_G \leq 180$)

angle_deg = θ_G - 90;

abscos = $\left|D\cos(\text{angle\_deg})\right|$

abssin = $\left|D\sin(\text{angle\_deg})\right|$

$ENU_{East-NewWypt}$ = $ENU_{East-A/C}$ + abscos;

$ENU_{North-NewWypt}$ = $ENU_{North-A/C}$ - abssin;

If ($\theta_G > 180$ && $\theta_G \leq 270$)

angle_deg = θ_G - 270;

abscos = $\left|D\cos(\text{angle\_deg})\right|$

abssin = $\left|D\sin(\text{angle\_deg})\right|$

$$ENU_{East-NewWypt} = ENU_{East-A/C} \text{ - abscos;}$$
$$ENU_{North-NewWypt} = ENU_{North-A/C} \text{ - abssin;}$$

If ($\theta_G > 270$ && $\theta_G \leq 360$)

angle_deg = θ_G - 270;

abscos = $|D\cos(\text{angle\_deg})|$

abssin = $|D\sin(\text{angle\_deg})|$

$ENU_{East-NewWypt} = ENU_{East-A/C}$ - abscos;

$ENU_{North-NewWypt} = ENU_{North-A/C}$ + abssin;

These procedures should then place the new "rabbit" waypoint in the correct spot to ploy the aircraft into adjusting for the real time wind.

3.5 – Wind Corrected Sensor Pointing

Assuming an efficient wind correction factor to the UAVs flight path, the aircraft would neatly track any predetermined waypoint-to-waypoint course. However, another wind related issue must be considered in order to provide a worthwhile attempt at real time wind correcting. The UAVs being exploited in the hostile, urban canyon environments are very small. Due to there size and payload restrictions any sensors, video or otherwise, must be equally small in both volume and weight. For this reason, most systems deployed on the aircraft do not have the ability to gimble the sensor head. Thus, even if the ground track of the aircraft is properly corrected, the UAV's nose will still "crab" into the wind. Therefore, the sensors would not be pointing forward, along the ground track, and would have the distinct possibility of not surveying the target, even if the UAV flew directly toward or over it, jeopardizing mission success. Thus, another

approach is presented that focused the wind corrections on the pointing direction of the on board sensors as opposed to the flight path of the UAV.

Viewing the UAV from the side, a right triangle can be constructed with the three sides being, the line of sight (LoS) distance for the sensor, the current altitude of the vehicle (Alt), and the horizontal distance the sensor projects (Horiz). Knowing the current aircraft altitude via the SDK, and assuming the sensor mounting angle, θ_{Sensor}, from the horizontal is known, the line of sight distance was determined, as is shown in Equation 36.

$$(LoS\_Dis)\cos(\theta_{Sensor}) = Alt \tag{35}$$

$$LoS\_Dis = \frac{Alt}{\cos(\theta_{Sensor})} \tag{36}$$

Now that the "LoS" and "Alt" variables were known, the horizontal distance that the sensor projected was found using Equation 38. As a check, with a θ_{Sensor} of 45°, the altitude and the horizontal distance should be the same value, and they are.

$$LoS\_Dis = \sqrt{Alt^2 + Horiz^2} \tag{37}$$

$$Horiz = \sqrt{LoS\_Dis^2 - Alt^2} \tag{38}$$

Next, the bird's eye view in Figure 15 must be taken into account in order to determine the appropriate offset for the UAV to fly.

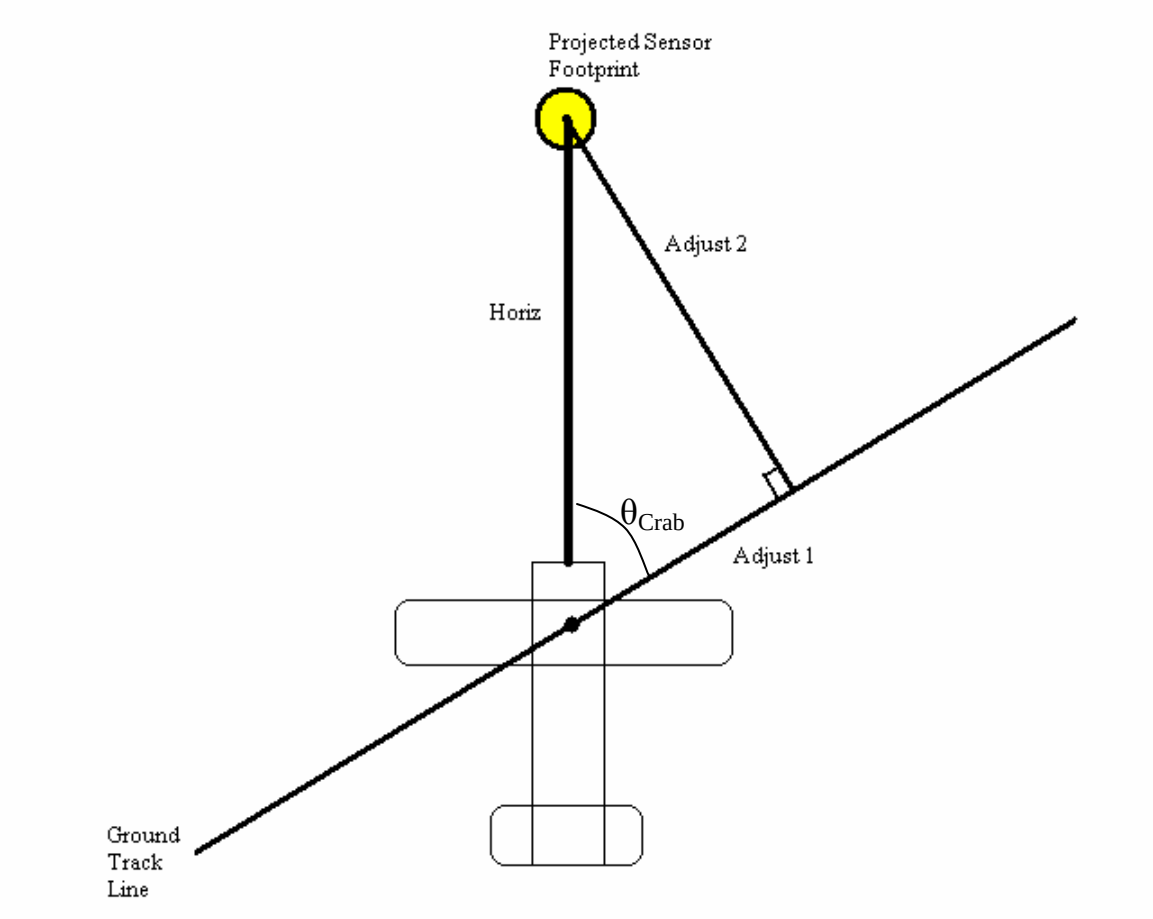

Figure 15. Top View of the UAV with the Adjustment Parameters Defined

The horizontal distance, "Horiz," now became the hypotenuse in a new right triangle as shown above. The other two sides of that triangle were the left/right (along the ground track) and up/down (perpendicular to the ground track) distances from the UAV to the sensor footprint. These two distances are the adjustments in the UAV's position required to put the sensor footprint at the current position of the UAV. These two values were represented as Equations 40 and 41.

$$\theta_{Crab} = \theta_G - \theta_{Mag} \tag{39}$$

$$Adjust\_1 = (Horiz)\cos(\theta_{Crab}) \tag{40}$$

$$Adjust\_2 = (Horiz)\sin(\theta_{Crab}) \tag{41}$$

Once these adjustments were known they would then be added/subtracted to the original waypoint/target ENU location; thus, providing an offset flight path that allowed for the sensors to survey the target, even under non-negligible wind conditions.

3.6 – Wind Correction Implementation

The implementation and integration of modifications onto an existing platform is a challenge equal to the development of the modification itself. Without proper integration, the entire project becomes purely academic. As with most real world projects, this process proved to demand the bulk of the man-hours invested in the research. On the other hand, the attempts at executing the wind corrections resulted in the majority of the useful research.

3.6.1 – Real Time Wind Estimating

The incorporation of the real time, updating wind estimation was fairly straightforward and successful. The Equations presented in section 3.2 were directly input into the C++ code with minimal issues. Because the Piccolo's telemetry packets were only used to passively read off information, the wind determination scheme was put into operation within a few days. Figure 16, below, is a screenshot example of the real time, updating wind estimates of a simulated UAV flight.

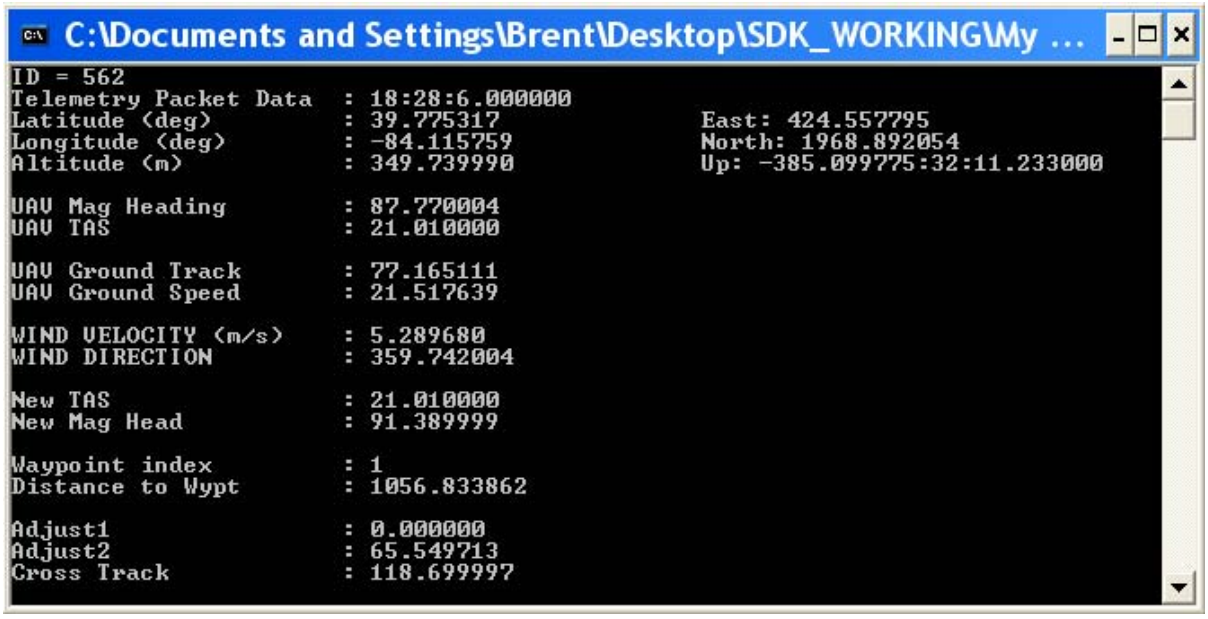

Figure 16. Screen Capture of the Piccolo SDK Executable

For most laboratory tests, the simulated wind input was 5 meters per second directly from the south. As will be shown, the results were within a reasonable precision (10%), especially when considering the simulation program induced random gusts. One primary concern with the wind finding code was the use of the arccosine math function used in Equation 14. Unfortunately, this function does not properly account for the sign conventions associated with the complete Cartesian coordinate system from 0° to 360°. Because of cosine/sine characteristics, if the data point was in the second, third, or fourth Cartesian-quadrants the appropriate applications of negative signs would not occur when implementing "arccosine." Fortunately, a two argument arctrangent function has been developed for math programming, called "atan2," which utilizes the proper sign characteristics of the tangent function throughout all four Cartesian quadrants. Therefore, Equation 14 was adapted to Equation 42, shown below.

$$\theta_W = \tan^{-1}(\frac{b}{a}) \text{ [rad]} \tag{42}$$

Variables “a” and “b” were the same as those in Equations 9 and 10, respectively. In the C++ code provided in Appendix B, this wind finding function is called “WindCorrection.” With this modification, the wind velocity and heading became a real time, viewable flight parameter that could be used to implement wind correction commands to the Rascal 110.

3.6.2 – Implementing the Turn Rate & Updating “Rabbit” Waypoint Approaches

Turn Rate Approach

A high proportion of time put into this thesis was spent attempting to implement these two approaches at wind correction for the UAV’S flight path. Essentially, both of the approaches attempted to modify the current UAV ground track to reduce its error in relation to the predetermined waypoint-to-waypoint path.

The first, turn rate, was to modify the aircraft magnetic heading, using updating turn rate commands, to directly affect the flight path. The basic idea was to directly command the new heading and TAS values at each time step. There was a time delay from when the wind affected the UAV to when the calculations and new parameters could have been uploaded back to the aircraft. However, with the request fast mode selected, this delay was under one second, which was considered negligible. This method would have then provided a close to real time heading and velocity adjustment. The obstacle then became sending the information back to the Piccolo II. Through this research, it was determined that the Piccolo II autopilot is initially uploaded with a set of waypoint data and then the system automatically attempts to fly the direct path

connecting subsequent waypoints. The system did not continuously send the waypoint information. So, when a new turn rate command was pushed through the system, via SDK, that command overruled all previous information and the aircraft only flew that turn rate. As an example, if the UAV was flying from waypoint 1 towards waypoint 2 at a heading of 270° and a command of 280° was required, the aircraft would be sent a turn rate command until the heading changed by 10°. Yet, instead of being able to command that 10° of turn and then returning to the predetermined flight plan, the operator would then have to continuously send turn rate commands; essentially, negating any waypoint tracking capabilities of the Piccolo II and attempting to fly the aircraft solely based off of turn rates. Now, aircraft control purely through turn rates has been proven to be a viable, and quite desirable, method. However, it was outside the intended scope of this thesis to alter the primary control method of the autonomous flight, but this topic may provide a worthy follow-on project as turn rate commanding carries with it numerous advantages.

Because of the known potential for progress in this area, the math and programming schemes required were kept in the attached SDK code. The mathematical background was formulated with the initial attempts at implementation represented. In addition to the "WindCorrection" function, the turn rate commanding algorithm utilized the "HeadingAdjust" and "AirspeedAdjust" functions. In "HeadingAdjust," the difference between the new, desired magnetic heading and the current magnetic heading provided the necessary adjustment. Then, this differential was divided by a time factor so that the turn rate command would not exceed a maximum rate of 20°/s. Finally, this rate was sent to the autopilot. With a completed algorithm, this command would be continuously updated, driving the aircraft to a correct heading.

As previously mentioned, a new TAS was determined from the "WindFinding" function as well. These alternate airspeed commands were successfully transmitted via the "AirspeedAdjust" function, located just below the "HeadingAdjust" function in the SDK. The reason the airspeed could be continuously updated was that the Piccolo II autopilot does not employ a time-based flight plan. The aircraft was only instructed to fly to a certain latitude/longitude location, altitude, with a specific airspeed, as opposed to intercepting a waypoint at a designated time interval. This allowed for the UAV to fly as fast or slow as aerodynamically possibly and for the operator to modify this flight parameter without interrupting the chosen flight plan.

"Rabbit" Waypoint Approach

Once it became clear that the turn rate approach was out of the scope of this research project, a second method of implementing real time wind correction was pursued. The method involved inserting an "updating waypoint" that was precisely placed such that if the UAV attempted to fly directly to this new waypoint it would actually end up at the original, desired target because of the wind effects. The new waypoint location would constantly be changing to counter variable winds and gusts. Additionally, the UAV would never actually reach the new waypoint, hence the name "rabbit." This enabled the operator to designate a distance from the original waypoint at which the "rabbit" function would be ceased, allowing the aircraft to initiate the switching logic to continue to the next predetermined target. Once the UAV was tracking the next waypoint, the "rabbit" would resume, repeating the process.

This approach afforded the desired result of a real time wind correction, but without having to alter the primary means of autonomous control (waypoint guided autopilot). Initially, the team was hopeful that this provided the solution. However, after implementing the algorithm in the SDK and running HITL simulations, it was observed that a key aspect of the Piccolo's operation prevented the efficient implementation. Cloud Cap's device was actually more of a flight path (track) follower than a true waypoint hunter. In a pure waypoint based system, the aircraft would designate where the target was located and then point the aircraft's nose directly at it, resulting in a Zermelo (Bryson, 1975 and Bryant, 1998) shaped path if wind was present. If the device had the capability to correct for wind, then a crab angle would be implemented and the UAV would fly a relatively straight path as long as the wind was constant. However, this is not exactly how the Piccolo II operated. It was established that the Piccolo II calculates a straight line path based on the position of the previous and next waypoints (its relative position and the position of the target). It then implements its own ground track algorithm in order to remain on that straight line path. Unfortunately, this algorithm was not as precise as would have been desired so an attempt was made to implement the above described wind correction artificially turning off the Picollo II'strack following mode and exploit a pure waypoint tracking method. When transmitting a new waypoint using the SDK, the operator was required to first set the waypoint location, and then send a second signal to track that waypoint. Examples, pulled directly from the wind correction code, of these C++ commands are provided below:

m_pComm->SendWaypointPacket(IDbrent6, &(newWPInfo), 69);

m_pComm->SendTrackCommandPacket(IDbrent6, 69, false);

The “SendWaypointPacket” command was fairly straightforward with its inputs being the physical autopilot identification number, a structure with the waypoint latitude, longitude, and altitude information, and then the desired waypoint number. The “SendTrackCommandPacket,” which sent the command to actually track the new waypoint, contained a twist with the “true/false” statement included as an input. The provided SDK “html” help files stated that “The third parameter (true/false) indicates if the vehicle should fly to the waypoint along the preceding track segment, or if it should go directly to the waypoint, using its current position as the starting point.” Thus, setting the parameter to “true” would command the UAV to go directly to the waypoint and a “false” would command the UAV to track the along the previous track in order to reach the new waypoint. At this point, the “true” setting appeared to be the solution, as the aircraft would fly directly to the new waypoint undergoing the effects of the wind and resulting at the original, desired location. However, after conducting tests with varying wind and waypoint locations it was determined that the Piccolo II software still created a direct path from the UAV’s current position to the new waypoint. So, the aircraft would employ its own ground track control in order to remain along that straight line flight path even though this was not clearly displayed through the Operator Interface. Unfortunately, this prevented further development of the concept. Thus, it was determined that the team could not “dumb down” the Piccolo II autopilot and have the aircraft fly a “Zermello” type flight path using the SDK. This was not to say that it would not be possible.

The two approaches presented above, turn rate commanding and updating “rabbit” waypoint, are believed to be completely valid methods for applying a robust wind correction algorithm to the Piccolo II autopilot controller. The math behind the

corrections provided for solid theory. However, due to the factors described, the team was not able to effectively implement these approaches. Without a working program to effectively adjust the flight path of the Rascal aircraft for real time winds the results of the research would have been paltry. Therefore, a completely new perspective was taken.

3.6.3 -- Wind Corrected Sensor Pointing

As aforementioned, if an aircraft is adjusting for wind or flying a straight line ground track in the presence of wind, then a crab angle is required for accurate navigation along a desired flight path. However, there exists a serious problem when these heading modifications are put in place. If the sensor gathering the information is situated such that it is pointed at a fixed angle off the nose of the aircraft and cannot gimble, there is a strong possibility that the sensor would never survey the target even though the aircraft flew precisely where it was supposed to. The small UAVs utilized in current operations have very little payload capabilities and can only carry a small, lightweight sensor system that will not be able to gimble. Thus, taking an alternate method to correct for wind, a set of updating and offset waypoints were calculated and then inserted and tracked such that the sensor was correctly pointed as discussed in section 3.5. In order to implement the new waypoints, a few modifications to the equations in section 3.5 were required.

Specifically, an angle, θ_1, was determined as the angle of the current track segment between the previous and next waypoints. This is shown in Equation 43.

$$\theta_1 = \tan^{-1}\left(\frac{ENU_{North-CurrentWypt} - ENU_{North-PrevWypt}}{ENU_{East-CurrentWypt} - ENU_{East-PrevWypt}}\right) \tag{43}$$

From this angle, it's complement was determined using Equation 44.

$$\theta_{Star} = (\frac{\pi}{2}) - abs(\theta_1) \text{ [rad]} \quad (44)$$

Equation 44 represented the transformation from the North=0° reference frame to the East = 0° frame. This value was then the angle at which the new waypoint was to be placed off of the original, assuming 0° was off the horizontal. The corresponding ENU east and north distances away from the original waypoint were calculated using Equations 45 and 46.

$$\text{sin\_from\_next} = -(Adjust\_2)\sin(\theta_{Star}) \quad (45)$$

$$\text{cos\_from\_next} = -(Adjust\_2)\cos(\theta_{Star}) \quad (46)$$

The reason for the negative signs was that the offset for the new waypoint had to be opposite in direction from the projected distances of the sensor. To then find the total ENU coordinates of the new, updating waypoint, Equations 47 and 48 were utilized.

$$ENU_{New-Wypt-East} = ENU_{Old\_Wypt-East} + \text{cos\_from\_next} \quad (47)$$

$$ENU_{New-Wypt-North} = ENU_{Old\_Wypt-North} + \text{sin\_from\_next} \quad (48)$$

For the purpose of allowing this new point to be continuously updated, the C++ function was written such that the above process would be repeatedly conducted as long as the UAV was within some distance, in meters, from the original target. The updating process is "turned on" for each waypoint when the ground distance between the UAV and the original waypoint was less than 400 meters and was "turned off" when the distance between the UAV and the new, adjusted waypoint was less than 100 meters. This logic to turning on and off the code was applied for two primary reasons. First, the "turn on" parameter allowed for maximum time and distance that the aircraft would fly along the

predetermined track. It was reasoned that the most time spent on track was desirable because of unknown factors off track. Additionally, the aircraft only needed to be adjusted in the final approach to the target in order for the sensors to capture that target. In a real world environment, to have the UAV fly off track for more time than was necessary would be allowing the introduction of more problems (e.g. collisions with fixed obstacles or detection by an enemy). At an altitude of 350 meters, which was where most tests were conducted, the sensor would project 350 meters in front of the aircraft if mounted at a 45°, which was the assumed angle for all testing in this thesis. Thus, the 400 meters criterion was chosen as the distance to begin the flight path modifications. The second reason dealt with the "turn off" parameter. At the point where the UAV was within a hundred meters of the new waypoint, the sensors would have already surveyed their target due to the field of view of the sensor. So, to avoid the "rabbit" situation described previously were the aircraft never actually reached the target, the code simply commanded the system to proceed to the original waypoint at that 100 meter mark. The SDK code accomplished this task by utilizing an "if/else" command on the "SendTrackCommandPacket" signal. The complete function is included as part of the SDK located in Appendix B.

As a note, all of the original flight plan waypoint information was "hard-coded" into the SDK. This does not provide for the best coding technique, but was required because the team was unable to capture the waypoint list and its corresponding data from the Piccolo's streams. However, "in the field" this may not be a complete disadvantage because the waypoints could be placed directly over any targets and the resulting latitude

and longitude information should be known. The operator could then simply append the code.

3.7 – Chapter Summary

This chapter provided a detailed look at the mathematics behind the three different techniques of wind correction evaluated during this research. Although the math and theory are believed to be solid, the implementation of that theory using the Piccolo II autopilot presented themselves as the road blocks. The two most conventional means at wind correction could not be implemented within the scope of this activity. However, the third, and operationally more significant, sensor pointing wind correction was successfully tackled and implemented.

IV. HITL Test Results and Analysis

4.1 – Overview

Chapter IV presents the results the research conducted during this thesis. Section 4.2 demonstrates the baseline ground track control capabilities of the Piccolo II autopilot system, the real time wind estimations developed in the previous chapter, and the corresponding ground position of the center of the sensor footprint. The three types of flight paths evaluated were a straight line point-to-point, a circular orbit, and the common racetrack pattern. Each of these was conducted with varying parameters. Section 4.3 displays the results of similar flight paths, but with flight path effects of the modified SDK code. Corresponding results from actual flight testing are presented in Section 4.4. The last section of the chapter (section 4.5) summarizes the results. As a note, it was assumed that the sensor was placed at a 45° mounting angle.

4.2 – Standard HITL Simulated Flight Tests with Real Time Wind Estimating

The most basic and essential flying characteristic for an aircraft is the straight and level flight path. Thus, the first simulation was a simple point-to-point flight path of three waypoints in a straight line. The simulated wind was set to 5 m/s from the south, almost a direct crosswind, while the UAVs commanded TAS was 20 m/s. These values represented a realistic flight condition with a moderate wind. The plot of this test is provided as Figure 17.

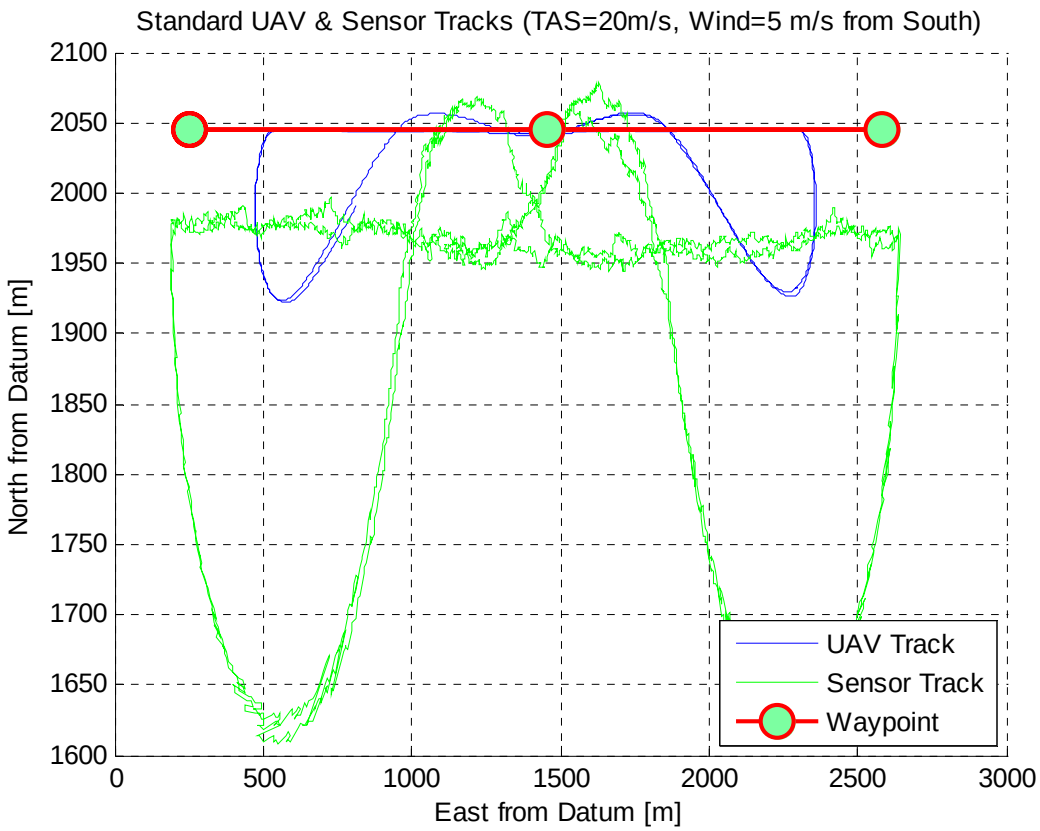

Figure 17. Standard UAV & Sensor Tracks for a Point-to-Point Flight Path

As shown and consistent with the technical discussions in Chapter 3, it became evident that despite precise ground track following, the sensor was tracking roughly 75 meters off of the desired position. The "crab" into the northerly wind, which results from the Piccolo II autopilot flying a straight ground track in the wind, caused the sensor footprint to be a significant distance off course.

Figure 18 and Figure 19 present various flight parameters corresponding to the previous graph. The speeds, altitude, magnetic heading, wind characteristics, and cross track distance were extracted off from the Piccolo's telemetry and then written to a data log using the SDK. The first four plots were primarily output as a "sanity check" for the

flight. It was pre-determined that most irregularities would be evident through observation and inspection of those four characteristics.

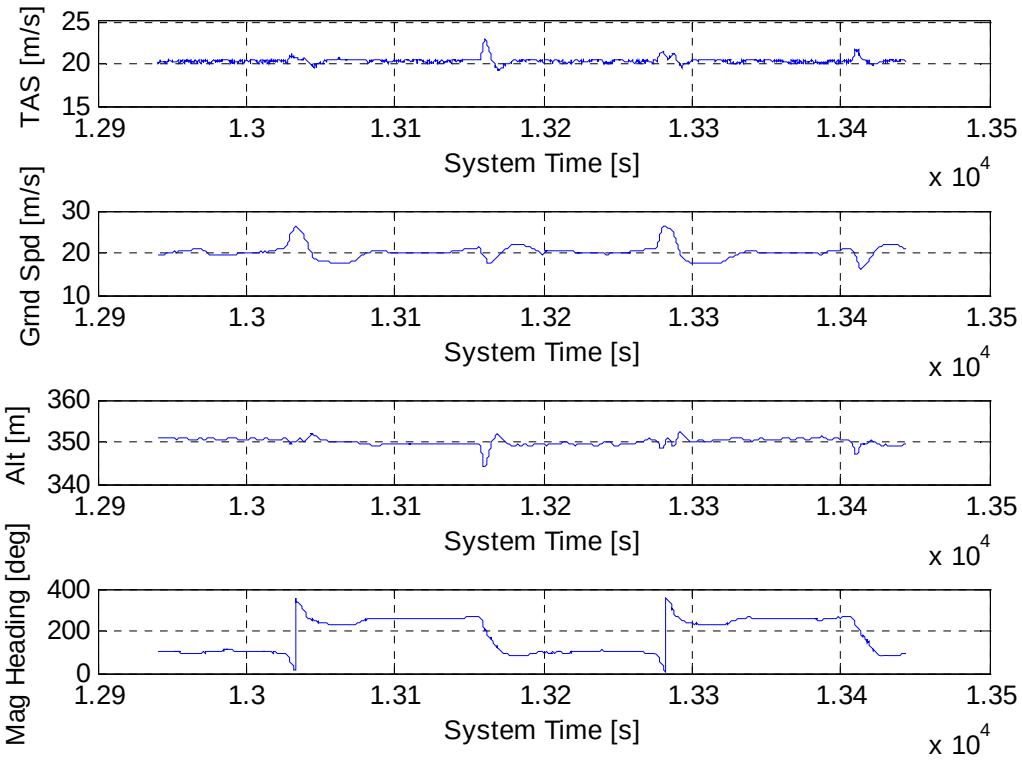

Figure 18. Various Flight Characteristics for the Standard Point-to-Point Flight.

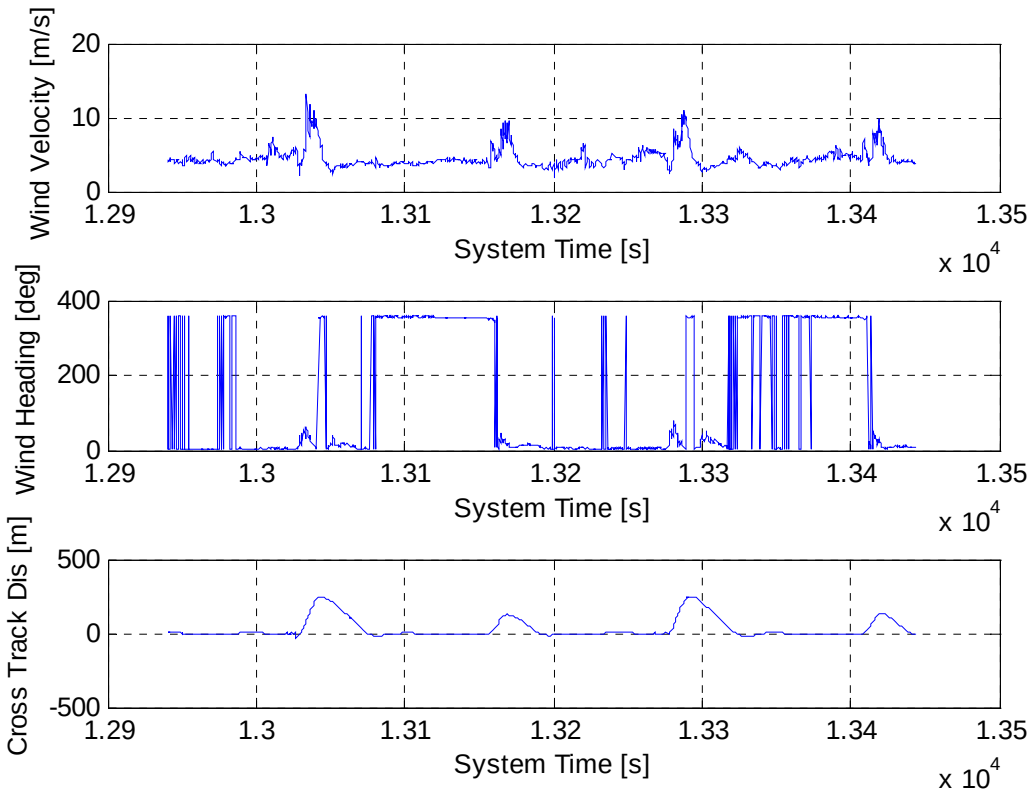

Figure 19. Wind Estimations & Cross Track Distance.

The real time wind velocity and wind heading estimations were logged from the SDK using the equations developed in Chapter 3. The wind characteristics in the HITL simulation were commanded directly from the simulation input values and were therefore considered constant (there was a turbulence setting, but this was kept at the "light" setting for all tests). However, the results from the updating wind estimations in Figure 19 were not always constant in either magnitude or heading. While these disturbances were not initially expected, the majority of the data still provided information of sufficient quality for a practical analysis. For instance, if the spikes were removed from the wind velocity plot, the average wind velocity was about 5 m/s. An analysis indicated the cause of the spikes. As the aircraft made large direction changes two issues arose: The first was that

the SDK calculations consistently lagged the actual aircraft position by one time increment. The second problem played off the first - as large heading changes occurred, the required “crab” angle would change at a significant rate. Because the code lagged behind the true position, when the computer caught up with the position it appeared as a large spike/step in that last transmission time period. Upon initial inspection, the wind heading plots appear to vary widely, but in reality they follow the same trend as the wind velocity plot. It is important to remember that a wind heading of 1° is essentially the same as a heading of 359°, validating the results. The airspeed as a function of time plot also displayed spikes. These were most likely due to significant heading changes as the UAV switched waypoints, and driven by rapid transitions from a head wind to a tail wind condition. The Piccolo II system simply cannot react instantaneously to such rapid changes and therefore there was an associated lag.

Figure 20, Figure 21, and Figure 22 depict the second test, which was a circular orbit about a stationary point at constant velocity and with a constant wind.

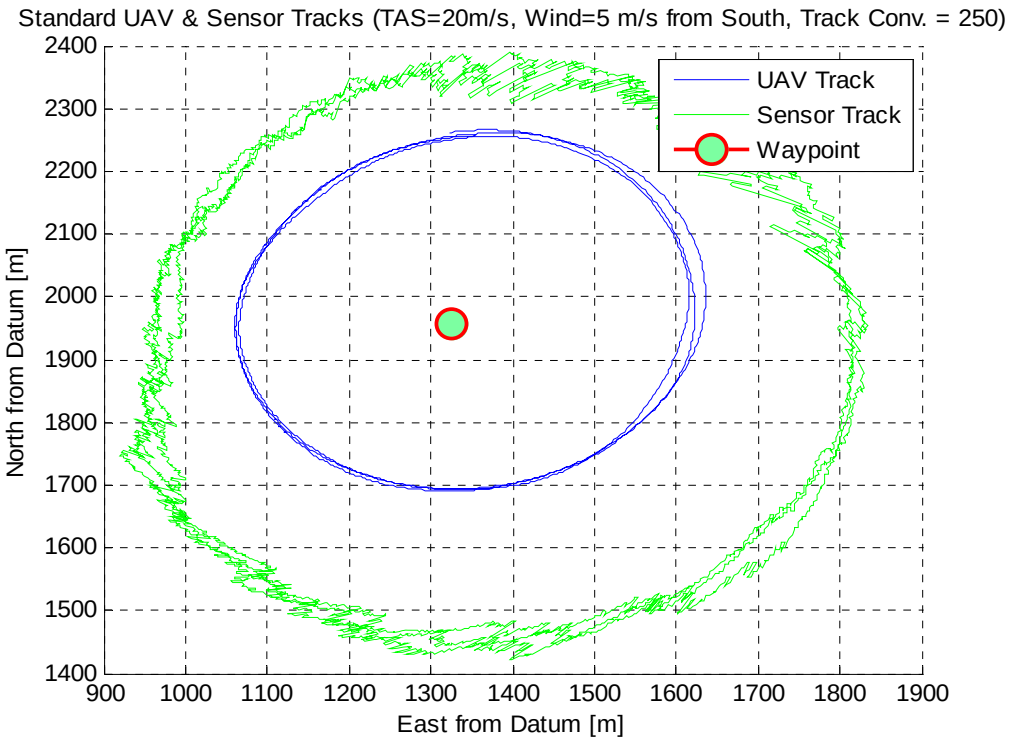

Figure 20. Circular Orbit Flight Path with Constant Velocity and Wind

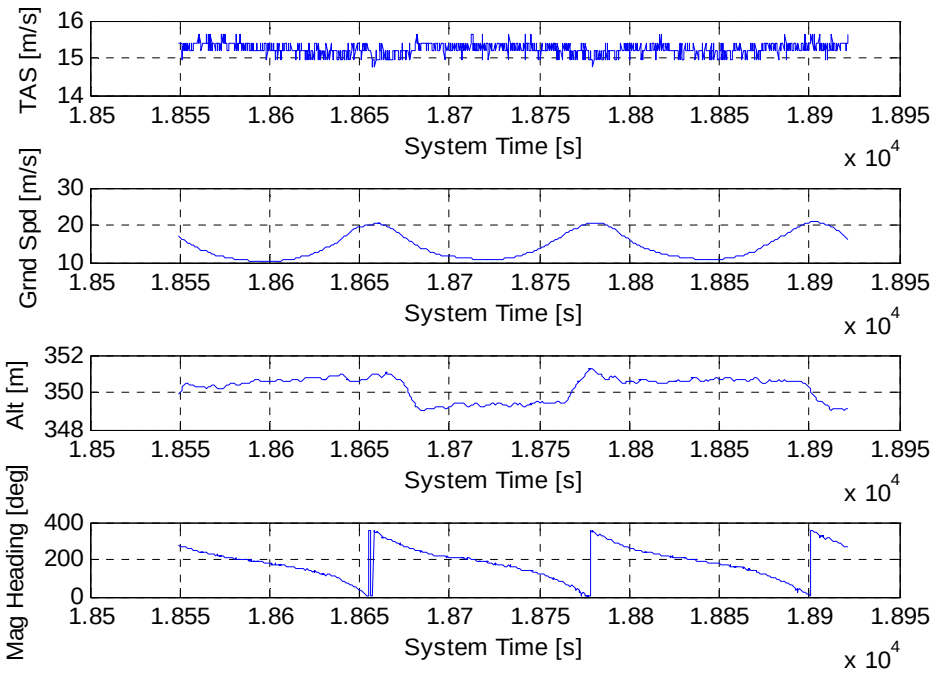

Figure 21. Various Parameters of the Circular Orbit Flight Path

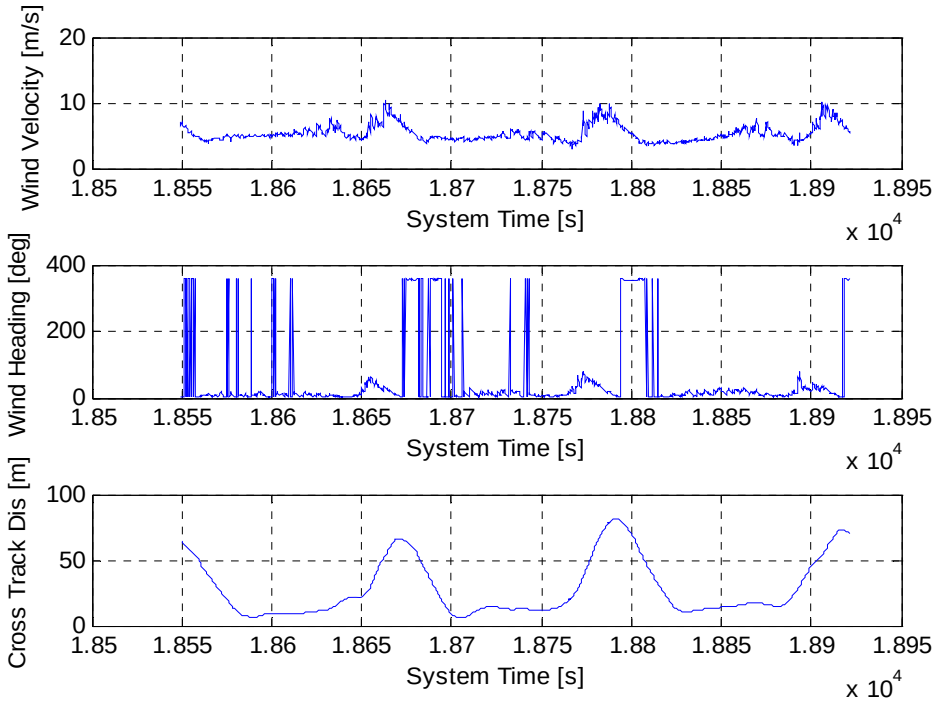

Figure 22. Estimated Wind Values for the Circular Orbit

The circular orbit flight path was interesting in that it displayed the Piccolo's bias when dealing with winds. As the winds were heading from south to north, it was evident that the UAV did much better when turning into the wind, i.e. incurring a headwind, as opposed to a tailwind. This was understandable as the ground speed would decrease and the aircraft would be able to better navigate at the slower speeds. The cross track difference between the head and tail winds was only about 50 meters. Having observed this, one must still recognize that Piccolo II manufacturer did a fairly good job considering this was a low cost, small scale COTS system. Yet, there were two things to consider when evaluating the overall performance. First, this was only a simulation, not the true flight characteristics and, second, with increasing commanded TASs, the cross track distance grew rapidly.

The following sets of plots depict the unmodified Piccolo II commanding the UAV in a race track pattern. At first the aircraft's velocity was the only parameter varied. Following those initial conditions, variations in the "Track Convergence" gain are presented. This gain drives the turn rate loop of the autopilot control software at the square of the velocity. Through previous research (Jodeh, 2006) it was determined that a Track Convergence gain of 250 appeared to be an optimal value for the Rascal 110 UAV. It will be shown that through lowering this value, the aircraft will attempt to stay on, and return to, the track with increasing aggressiveness. However, the faster convergence did come with a loss of precision of altitude hold due to more aggressive turning and banking of the UAV. Figure 23, Figure 24, and Figure 25 present the results of the standard autopilot commanding the predetermined racetrack pattern at TAS=12m/s, with a wind of 5 m/s from the south, and Track Convergence (TC) =250.

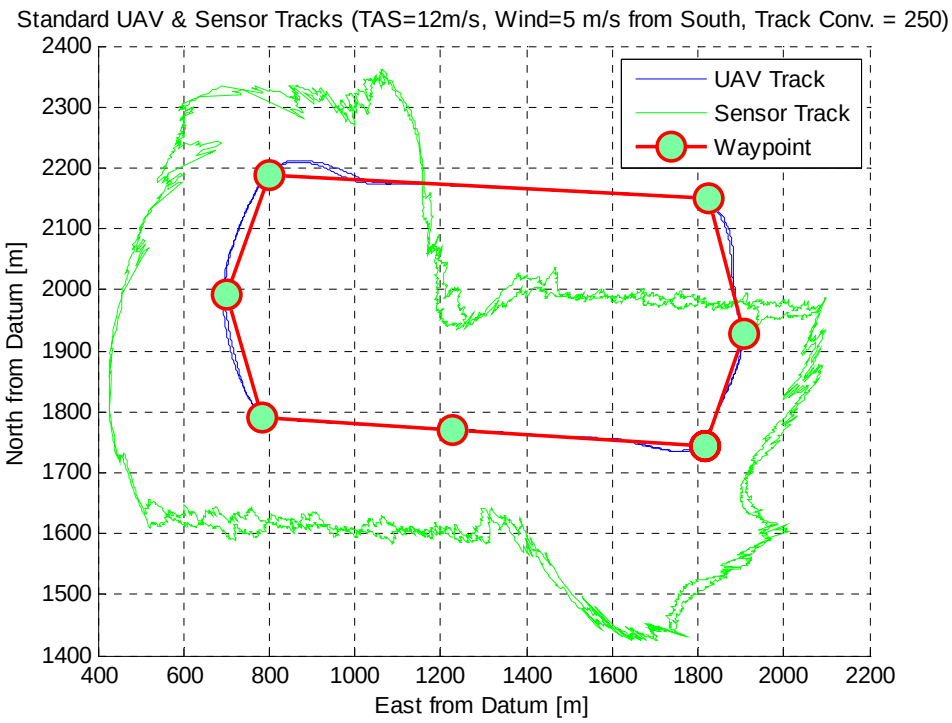

Figure 23. Race Track Pattern with TAS=12m/s & Wind= 5m/s

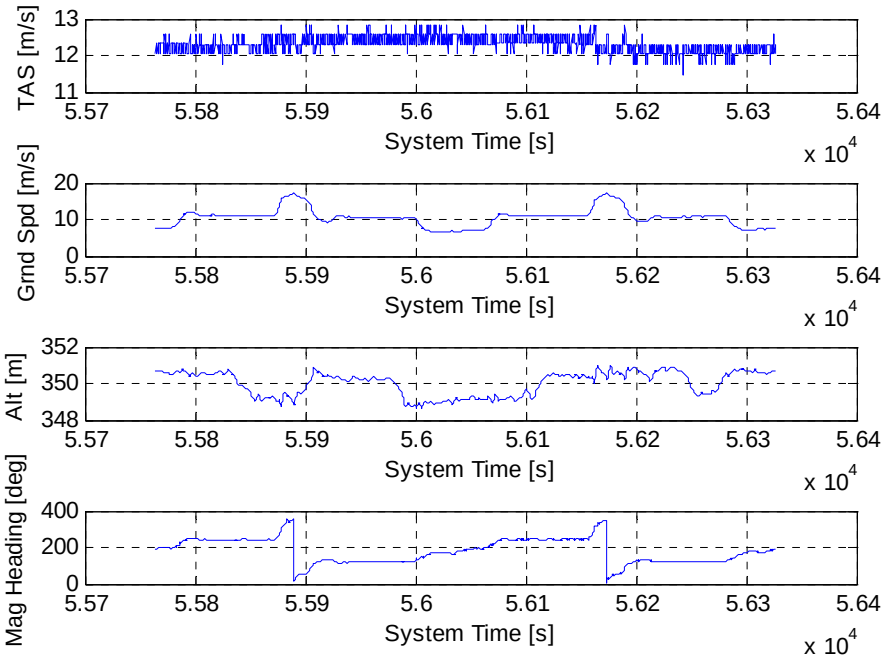

Figure 24. Various Parameters for the Race Track Pattern at 12m/s and TC=250

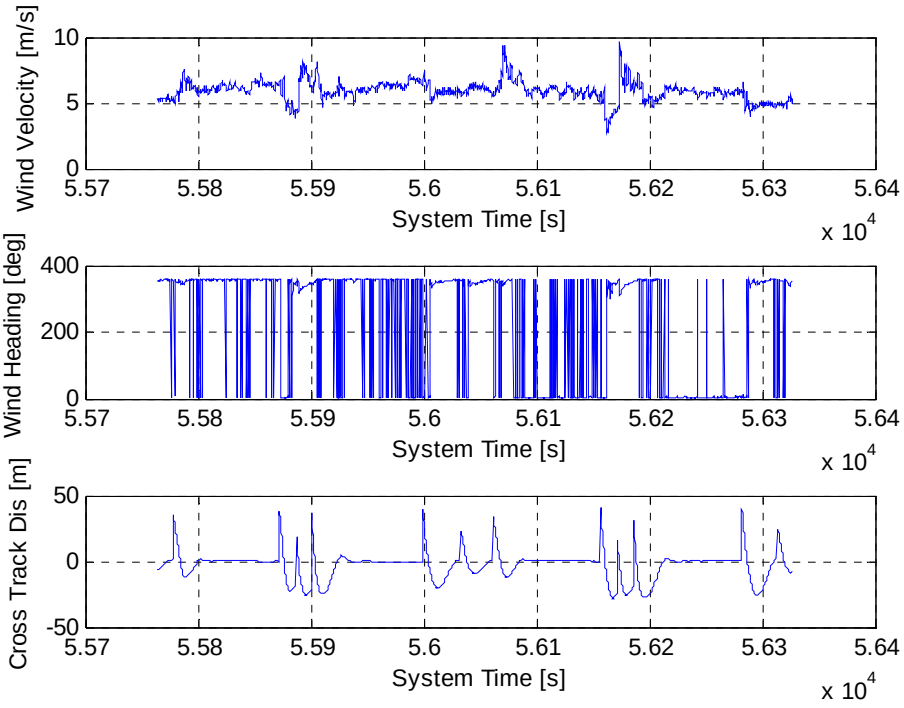

Figure 25. Wind Estimations & Cross Track Distance

The sensor paths plot revealed that even at the slowest operating speed of 12m/s, the sensor footprint would remain between 100 and 200 meters off of the ground track. Fortunately, the physical aircraft tracked the desired path extremely well with maximum cross track values of less than 50 meters. This appears sufficient for the urban canyon flight regime. Again, the estimated real time wind values provided adequate depictions of the current flight conditions.

The next series of tests were identical to those just described but with variations in the true airspeed (TAS). In addition to the 12 m/s run, 15 m/s, 20 m/s, and 30 m/s evaluations were conducted utilizing the same “race track” waypoint locations.

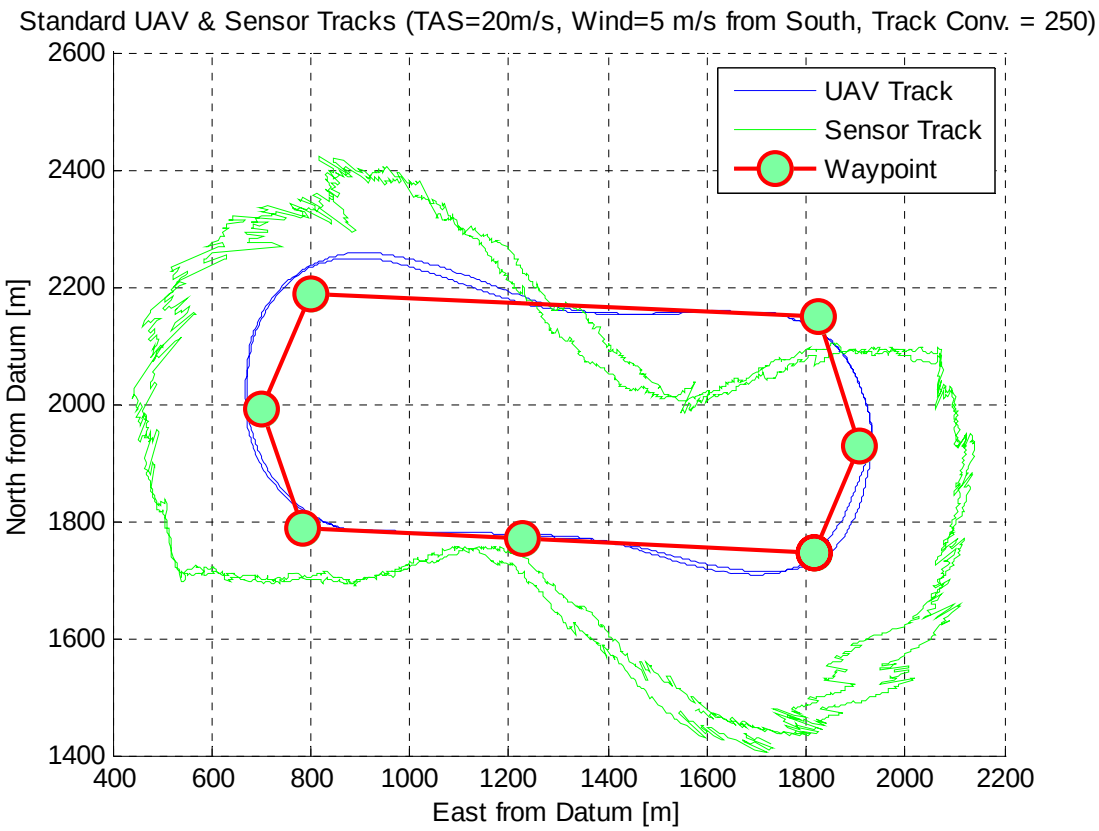

Figure 26. Race Track Pattern at 20 m/s Track Conv.=250

As expected with the higher velocity, the small aircraft was less capable of precisely holding the track as shown by the blue line in Figure 26. As a result, the sensor footprint tracked further off course. Any close contacts with the waypoints and the sensor track were purely coincidental and would not have occurred with differently spaced points. With the track convergence gain set at 250, 20 m/s was about as fast as the UAV could fly any semblance to the race track shape. As shown in Figure 27, at 30m/s an oval was the best the aircraft could accomplish. However, if the race track had longer distances between each waypoint the Rascal should have been able to fly an acceptable pattern. As a baseline test, this provided strong evidence that with a relatively small pattern and a nominal wind, the aircraft could not be relied upon fly a precise track.

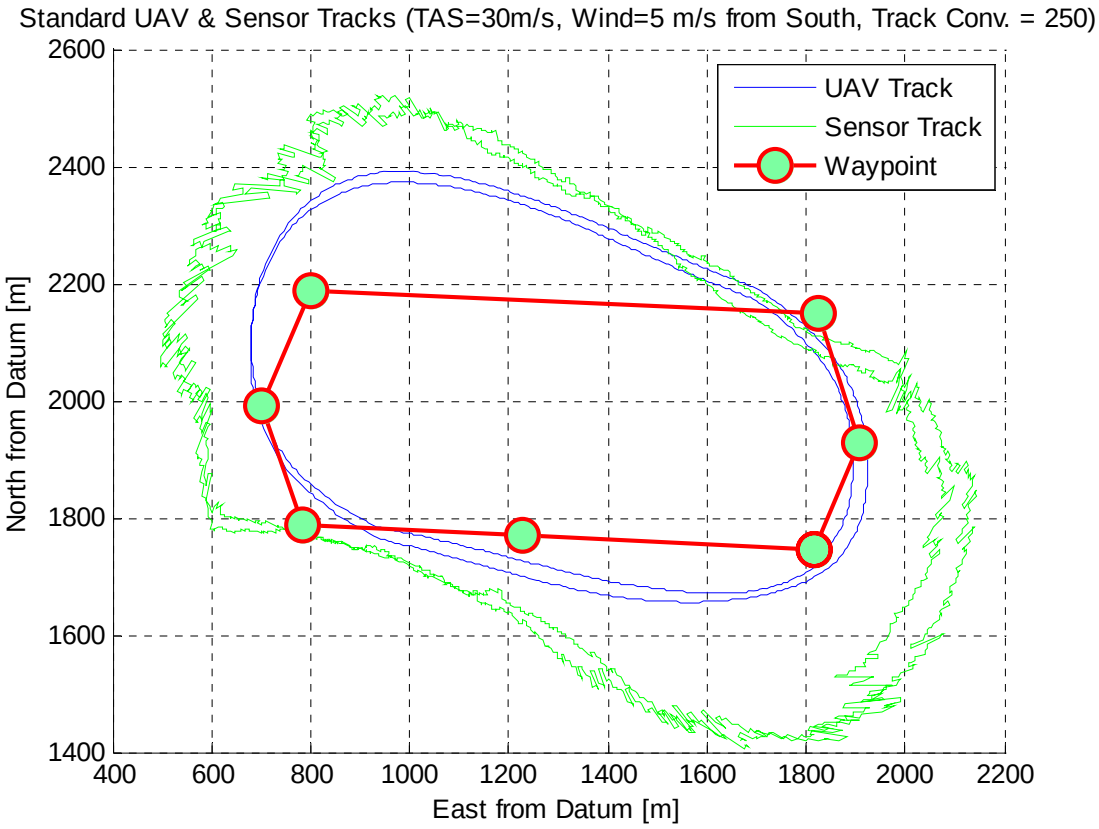

Figure 27. Race Track Pattern at 30m/s with Track Conv.=250

Because of the issues described above, the remainder of this document will focus on the 12 m/s and 20 m/s cases. Additional results can be found in Appendix A. These two airspeeds correspond to two crucial flight situations. The 12 m/s runs represented the best results and the 20 m/s evaluations were consistent with a common actual flight condition.

In an attempt to acquire improved results, the track convergence (TC) gain was reduced to a setting of 150 and then to 50. The weighted importance of flying the straight line track between two subsequent waypoints would be increased while the smoothness of that track and possible altitude criteria would be lessened. The TC variation plots at 12 m/s will be presented first followed by the corresponding results at 20 m/s.

At 12 m/s, Figure 28 displays that the Piccolo II did a very good job at remaining on track. However, an interesting side effect began to appear. With the lower gain value for track convergence the aircraft appears to bounce between some designated cross track bounds, similar to a bowling ball going down a lane with bumpers. This was shown by the sensor position beginning to waiver left and right, especially along the longer straight segments. Subsequent figures will bring this side effect into a clearer view. The cross track distances for the respective 12 m/s runs decreased from a 40 meter maximum to about a 25 meter maximum. For the urban canyon flight regime initially investigated for this thesis, such a simple adjustment to the Piccolo II autopilot created a significant increase in the track following performance.

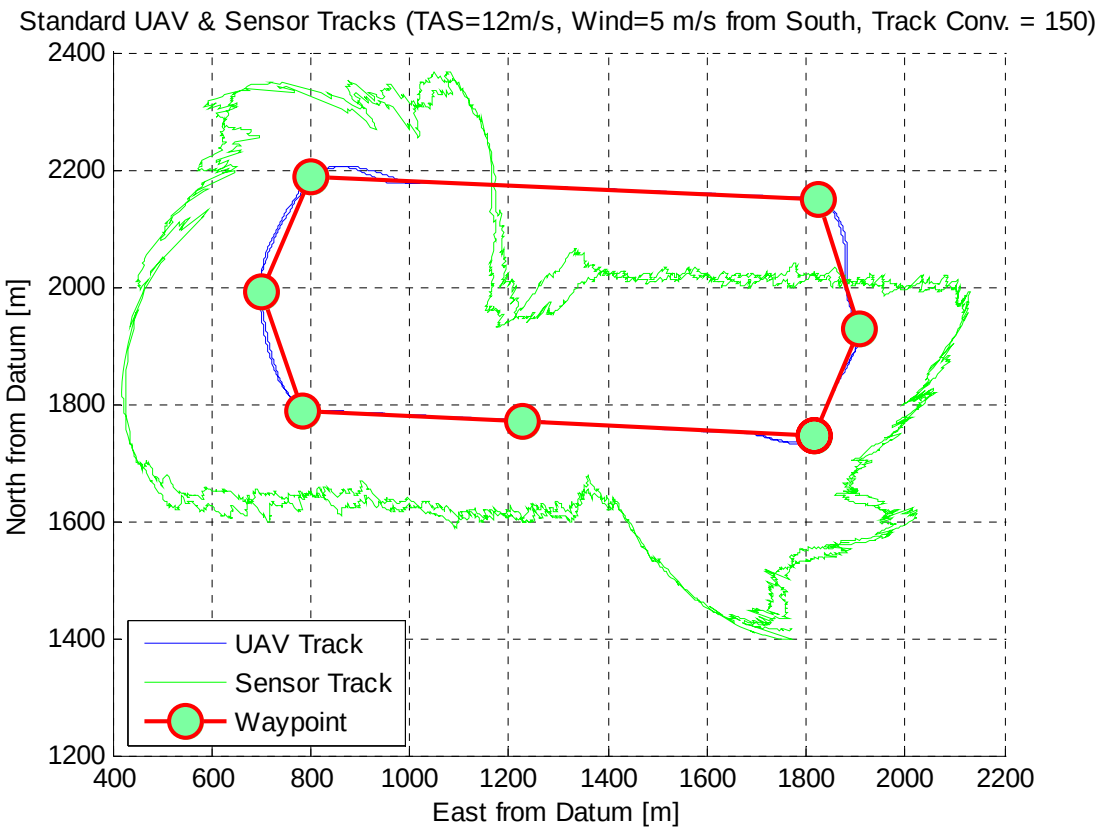

Figure 28. Race Track Pattern at 12 m/s with Track Conv.=150

Figure 29 was the 12 m/s run at a track convergence gain of 50. This time the blue line representing the actual aircraft's position can barely be seen as it is coincident with the desired track for most of the flight. However, this "scanning" side effect became excessive. The nose of the aircraft was continuously moving laterally in an attempt to remain as close to the track as possible. Once again, any points at which the sensor footprint and the targets were close were coincidental. This result would not be acceptable for actual flight.

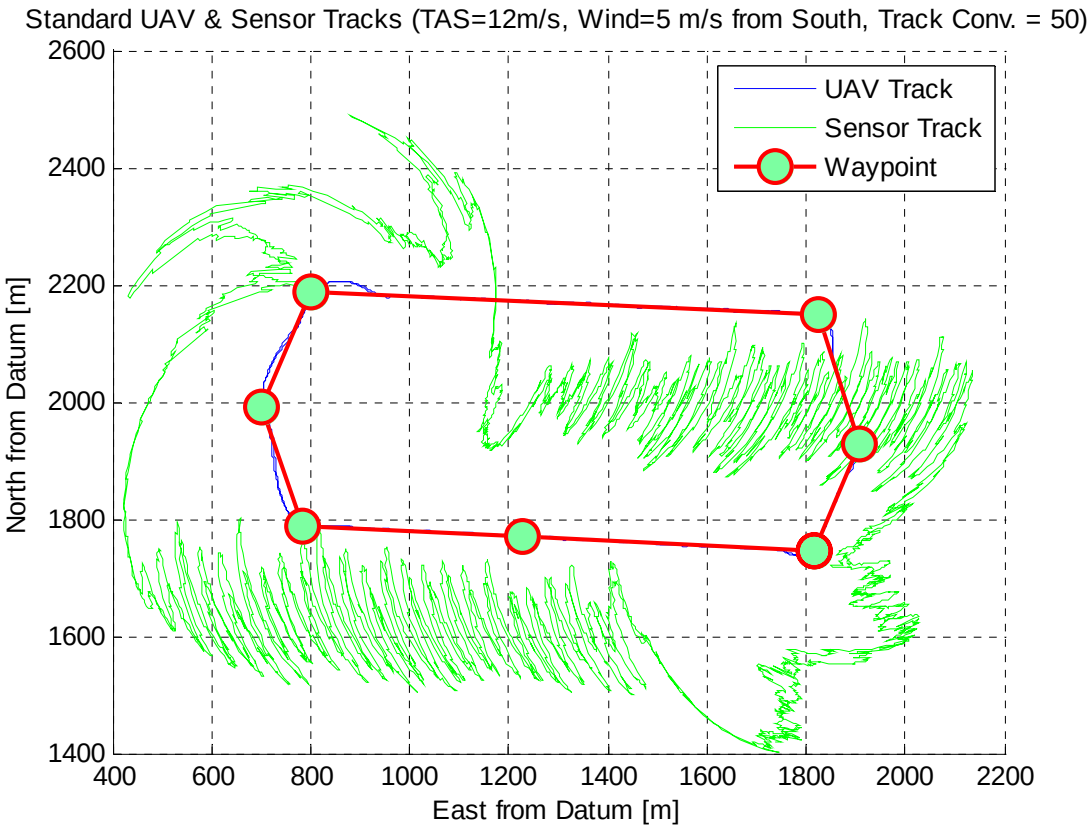

Figure 29. Race Track Pattern at 12 m/s with Track Conv.=50

The 20 m/s run with the track convergence set at 150, Figure 30, showed the expected decrease in tracking ability when compared with the 12 m/s, but an improvement over the respective 20 m/s run with the gain set at 250. The quicker response to return to the track was the most notable change. Because the track holding was improved, the sensor position better mirrored the track, but the offset was still present due to the crabbing. The sensor track was also beginning to become jittery, but not so drastic as to render the condition useless.

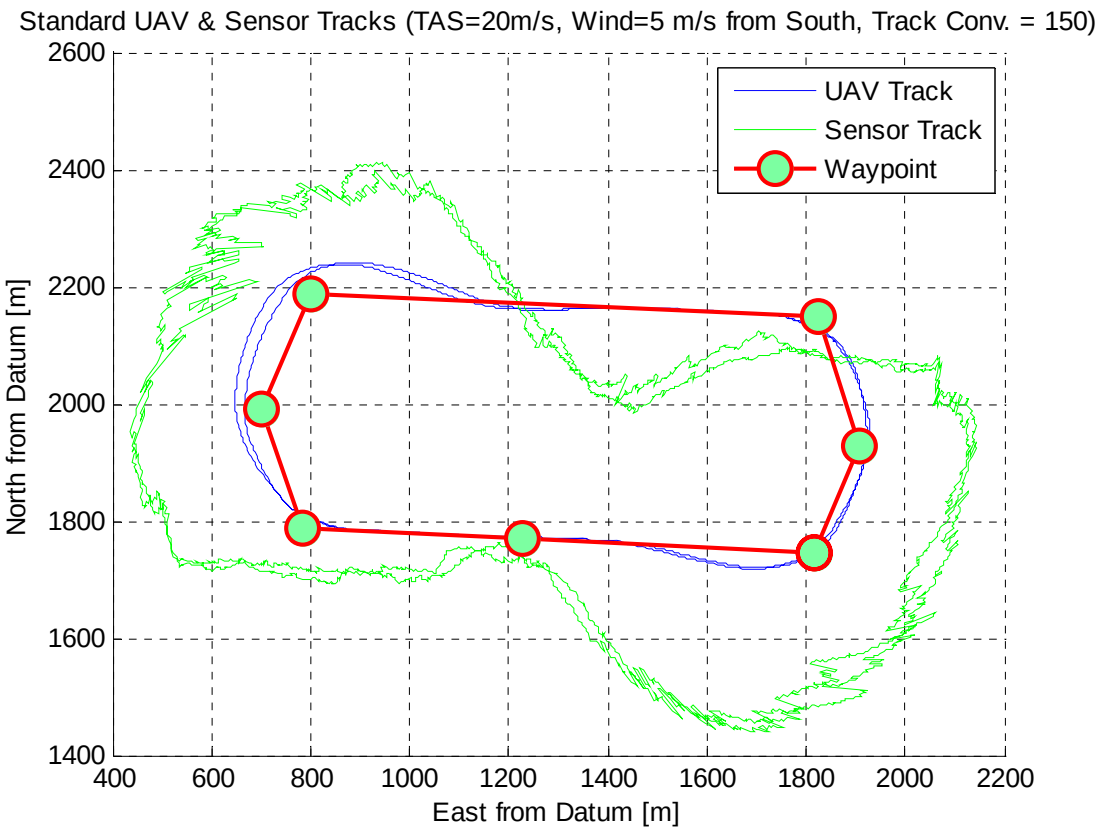

Figure 30. Race Track Pattern at 20 m/s with Track Conv.=150

With the track convergence reduced again to 50 in Figure 31, a slight improvement in the UAV flight path was observed. However, that small improvement was outweighed by the increased sensor waiving. It is important to notice that despite the decreased tracking performance as compared to the 12 m/s run, the "induced scanning" was not nearly as prevalent. The reason for this was that because of the higher velocity, the aircraft was not as susceptible to the wind. With a wind of 25% percent of TAS as opposed to 41.66% as with the previous runs, the UAV was able to better handle the aerodynamic forces as the increased velocity would effectively increase the control powers of the rudder and ailerons.

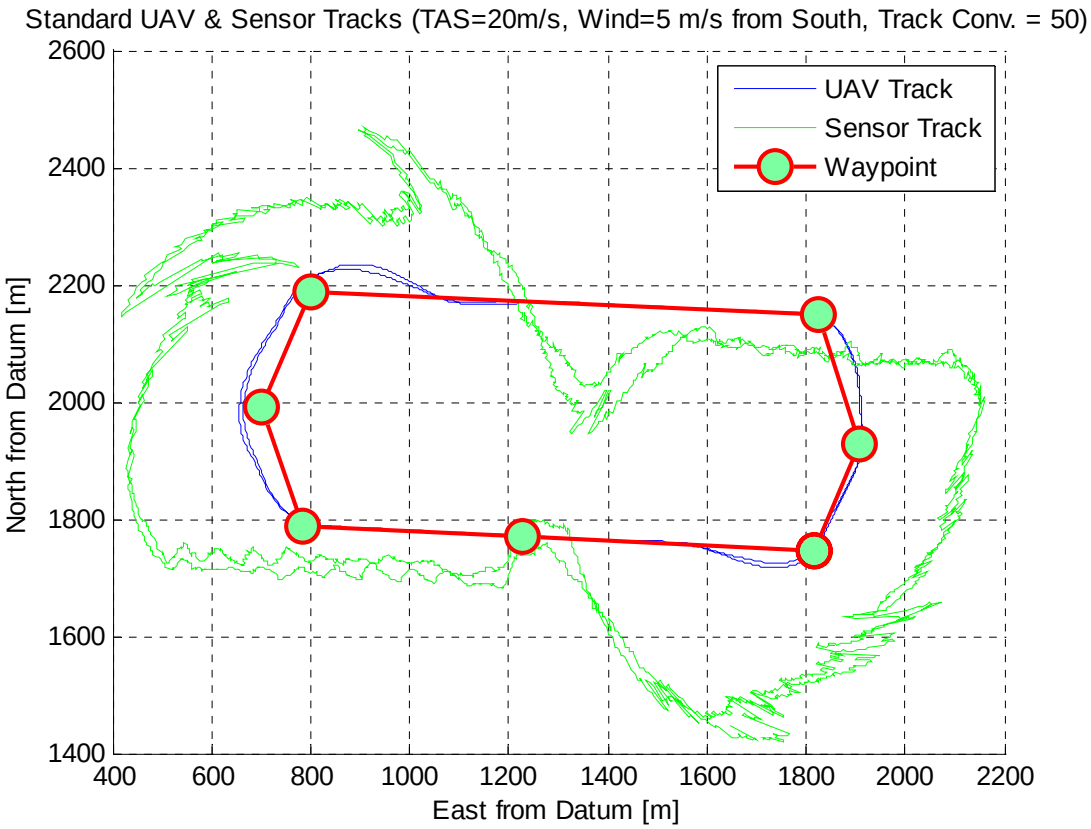

Figure 31. Race Track Pattern at 20 m/s with Track Conv.=50

The entire set of baseline tests provided insight into two key objectives of the research; the real time wind finding results and the sensor pointing issues. The results of the real time wind finding were considered a success. Despite a few points when the wind velocity and/or direction would spike, the results were consistently accurate under various operating conditions and flight paths. Utilizing the wind finding algorithm in the SDK, a passive procedure was provided that allowed for simple means to view and then log the wind data, along with numerous other telemetry variables. The results would be best utilized as a situational awareness aid or to post process data for future test flights. The full set of results is supplied in Appendix A.

The second set of pertinent data concerned the location that a nose mounted sensor would actually be pointed when the UAV was in the presence of winds. From the bird's eye views of the SIG Rascal's simulated flight path, it became clear that the sensor's footprint would not survey the desired target (waypoint). The tests conducted at the slowest speeds did hold the track the best, but the aircraft required a crab angle to accomplish that task; thus, resulting in a lack of coverage of the target by the sensor.

4.3 – HITL Simulation with Wind Correction

4.3.1 – Turn Rate & Updating "Rabbit" Waypoint Approaches

As previously mentioned, the turn rate and "rabbit" approaches of track following improvement were not successfully implemented on the Piccolo II autopilot. However, the time spent on researching these two possibilities did return some useful results. First, turn rate commanding was, and still is, a feasible means for wind correction. In the long run, this is probably going to be the best and most accurate means for wind correction on small UAVs. Second, the "rabbit" waypoint chasing would be an acceptable means of real time wind correction, with the added advantage of being easier to implement into the Piccolo's SDK or any waypoint guided autopilot. This "rabbit" chasing algorithm is implemented in the C++ code provided in the appendix – and works for a single waypoint. Accessing the "list of waypoints" from the SDK would enable full implementation. Once this Piccolo II specific issue is resolved the rest of the correction algorithm should be simulated in C++ code.

4.3.2 – Wind Corrected Sensor Pointing

Using the same predetermined flight paths as in section 4.2, a direct comparison was made to determine the effectiveness of the implemented wind corrected sensor pointing. This algorithm used the SDK to actively modify the flight path of the Rascal in the HITL simulation in an effort to induce an offset that allowed the simulated on-board sensor to survey the target. For the research, it was assumed that, operationally, a waypoint would be set directly over any target.

Figure 32 depicts the same straight line path as in Figure 17, but this time the SDK code was actively placing a new waypoint at a calculated, ENU distance away from the original. The graph also connects the corresponding positions between the center of the sensor footprint and the aircraft. Under the same flight conditions, the center of the sensor footprint was, at best, 75 meters from the waypoint. As shown in Figure 32 below, this error was reduced to about 10-20 meters when the wind correction was employed. For this flight condition, that was about a 75% reduction in error. The updated waypoints clearly provided the necessary corrections so that the sensor could inspect the target. Additionally, because the code was designed such that the aircraft would remain on track as long as possible and then jump out to capture the target, there were no radical direction changes which would have caused drastic elevation changes. Just as with the results in section 4.2, the complete set of plots is attached in the appendices.

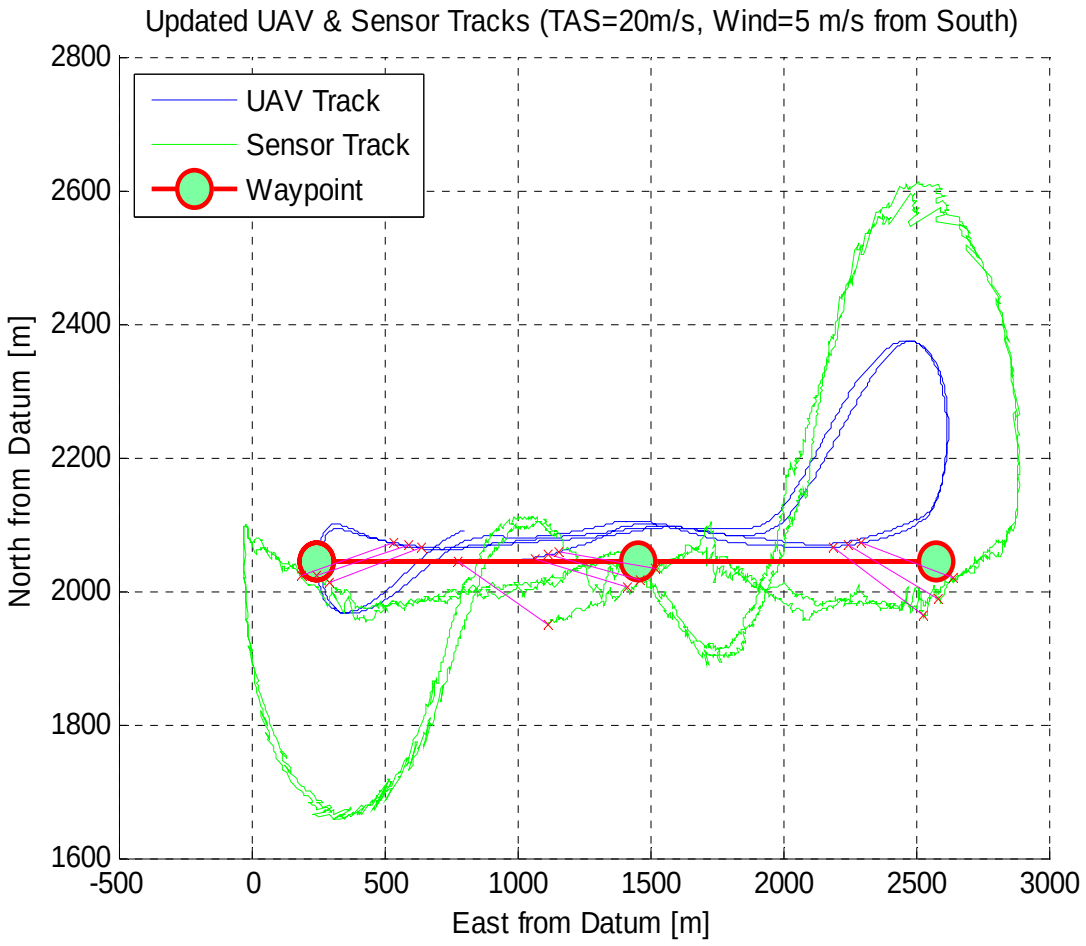

Figure 32. Point to Point at 20 m/s - Adjusted for Sensor

The straight line, point to point flight track was used as an initial proof of concept and that the modifications could be implemented efficiently. The more important, and realistic, test was to implement the code on the race track pattern. This would evaluate whether or not the new waypoints would be placed correctly given a varying relative wind. The track convergence gain was set to 250 for all of the simulated tests involving the waypoint adjustments. The reason for this was that the "induced scanning" could possibly introduce significant errors in the crab angle calculations. As a note, this gain

could have been increased in an attempt to smooth out the track, but this was not evaluated.

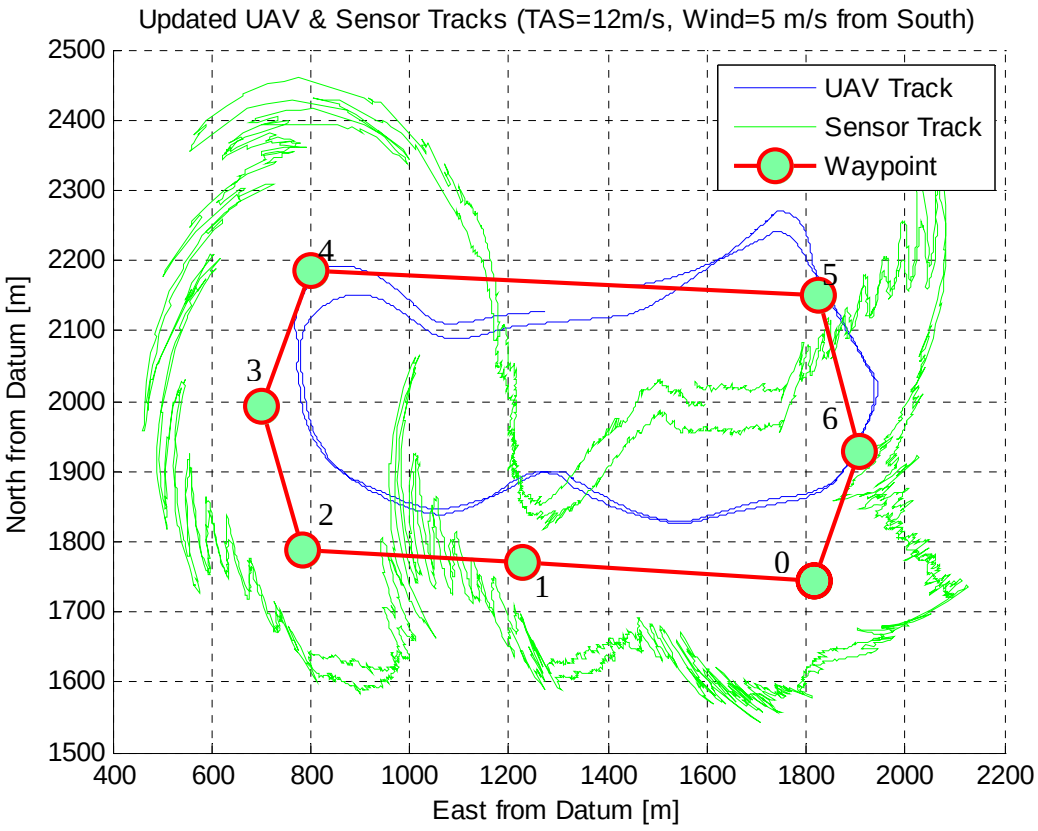

Figure 33. Race Track Pattern at 12 m/s - Adjusted Waypoints

Figure 33 is a plot of the results from the race track pattern at 12m/s TAS and the wind of 5 m/s from the south. The error distance between the sensor footprint and the waypoints was decreased for most of the targets. However, the jittery sensor path was unexpected. The scanning effect, which was attempted to be avoided by using the convergence gain of 250, was observed. It was conjectured that this occurred because of the continuously updating waypoints. At each time step the algorithm updates the placement of the waypoint. So the waypoint will move slightly left/right, up/down. Thus,

with the waypoint moving slightly, the aircraft needed to adjust its heading at each time step. This resulted in the "induced scanning" effect.

Starting with waypoint 0 at the bottom right of Figure 33 and counting clockwise, the results of the wind correction for waypoints 1, 2, and 5 were quite favorable. These three all had an error of less than 50 meters. Waypoints 0 and 6 had marginal results with about 100 meters of error. Waypoints 3 and 4 did not have improved results when compared to the standard Piccolo II. They were not any further away, but the scanning effect would be undesirable. The tail wind condition encountered as the vehicle turned towards waypoint 3 coupled with the small track segments proved to be too much for the Rascal as it was not able to navigate the right hand turn while incorporating the sensor pointing offset. Longer track segments would have resulted in much better results as the aircraft would have steadied itself on track before attempting to implement any modifications. The head wind condition produced closer distances as explained previously.

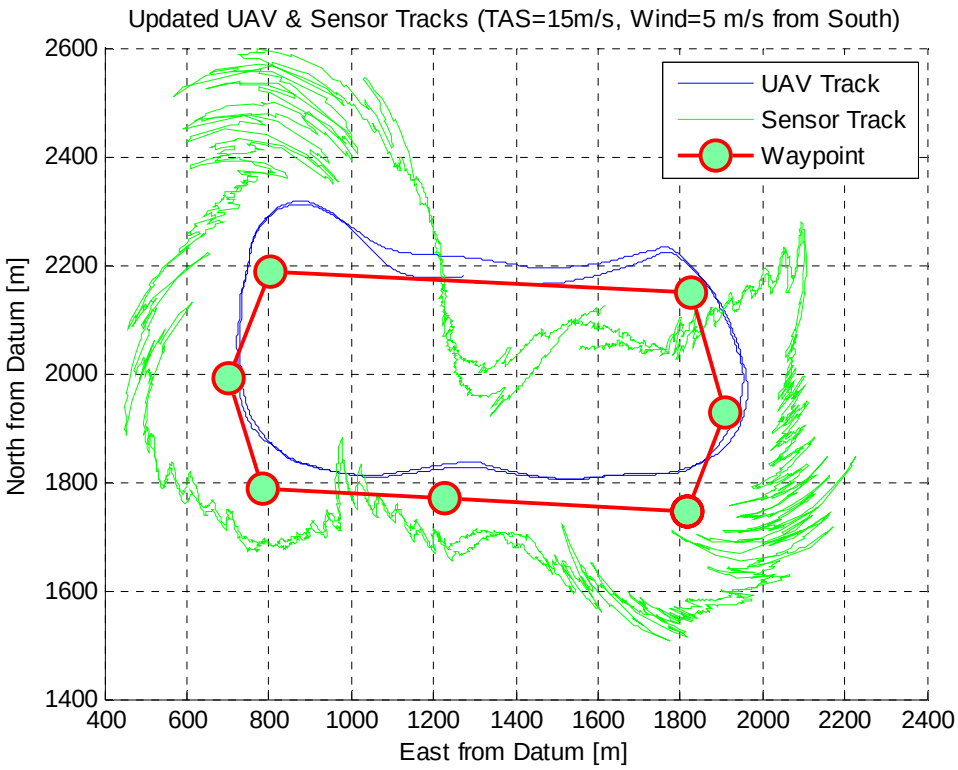

Figure 34. Race Track Pattern at 15 m/s - Adjusted Waypoints

As the TAS was increased in Figure 34, Figure 35, and Figure 36, the results mirrored the unmodified tests with a reduction in the wavering effect and a gradual reduction in track following precision. The greatest improvement remained with waypoints 1, 2, and 5 as they were still the longest track segments. In Figure 36, the resulting UAV track was actually improved over the unmodified test at 30 m/s. Overall, the data for the race track pattern were mixed. There were significant improvements in the sensor footprint error for approximately half of the targets, with the other half having only a marginal or no improvement. However, it was determined that if all track segments were of sufficient length the results would have been more desirable throughout.

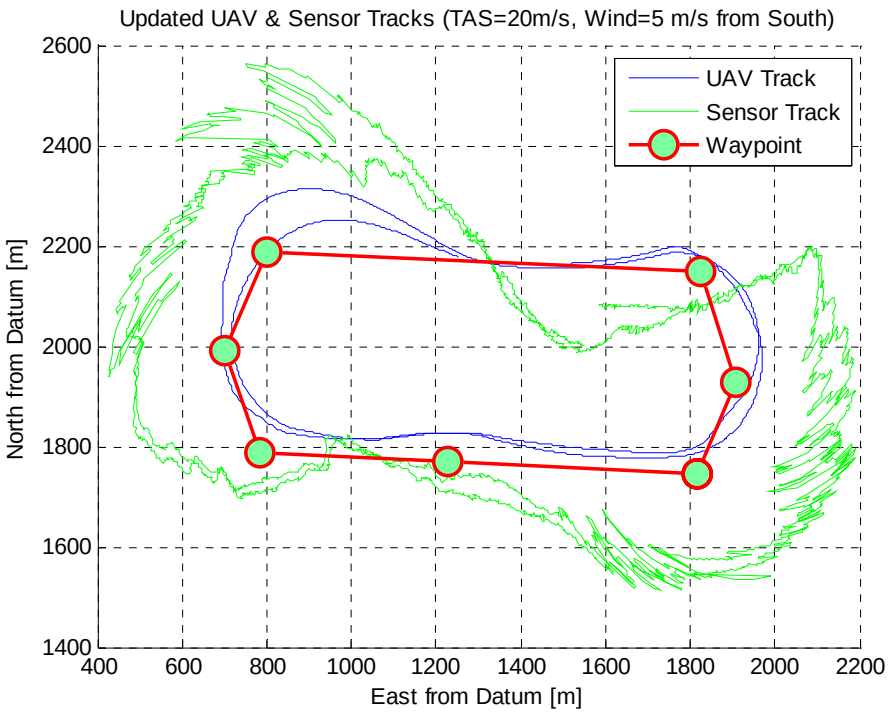

Figure 35. Race Track Pattern at 20 m/s - Adjusted Waypoints

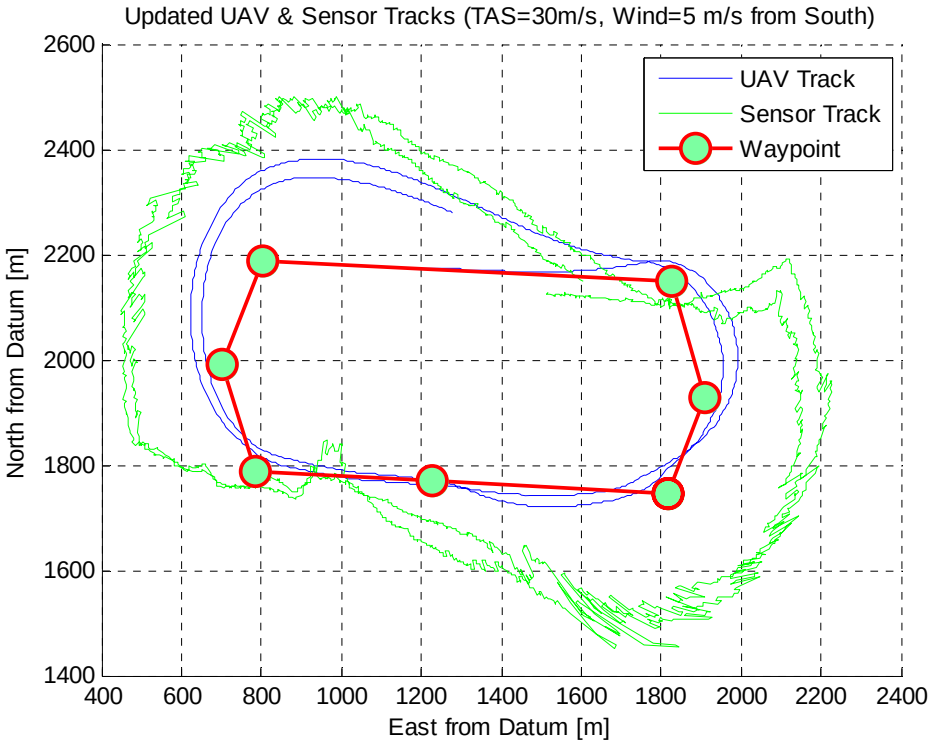

Figure 36. Race Track Pattern at 30 m/s - Adjusted Waypoints

Varied Environmental Conditions Tests

To ensure some level of robustness in the sensor pointing code, two additional evaluations were conducted. The first varied the small UAVs altitude. Because the correction distance was based upon the distance between the center of the sensor footprint and the aircrafts location, varying the altitude would vary the forward, lead distance of the sensor footprint. Figure 37 is the graphical representation of this test.

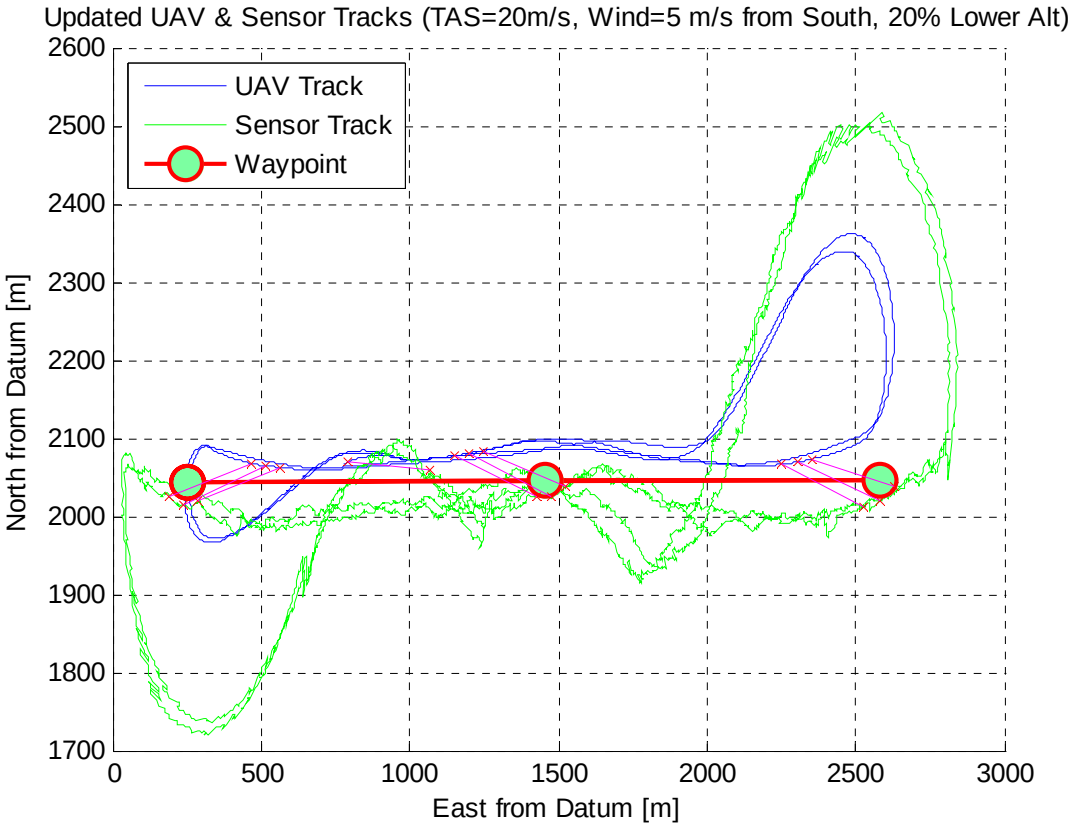

Figure 37. Point to Point at 20 m/s and 20% Lower Altitude

The Rascal's lower altitude would mean that the sensor would not be projecting as far ahead of the aircraft. For this reason, the required offset distance for the new, updating waypoints should be less. Figure 37 clearly shows that the offset distances were less

drastic and as a result the sensor path actually comes closer to the targets. For this test, the average miss distance was less than 20 meters. Based off these conclusions, it was assumed that if the UAV's altitude was increased that the new waypoint offset distance would have been increased.

The second additional test returned the aircraft to the previous 350 meter altitude criterion, but doubled the wind velocity to 10 m/s. Also, the direction of the wind was switched 180° to a heading of due south. The outcome, as presented in Figure 38, showed a reversal of offset direction in addition to an increase in the required offset distance. These results displayed that the algorithm had the capability to make the appropriate adjustments based on a current wind velocity and direction.

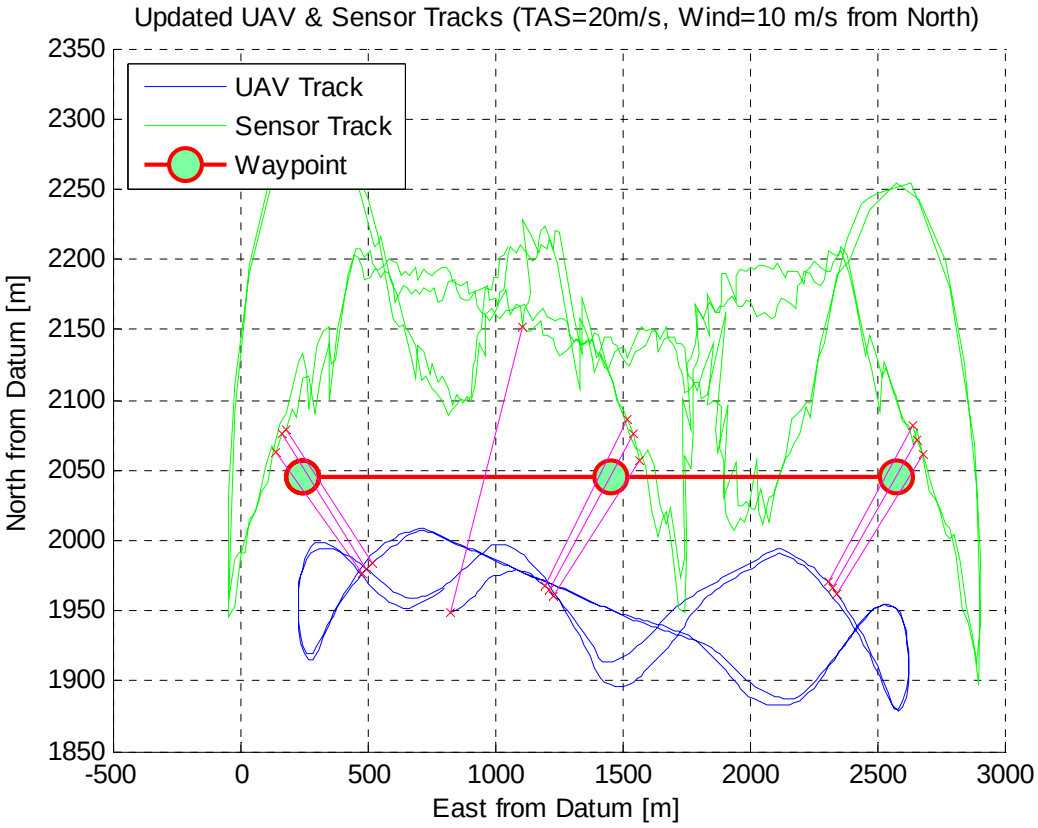

Figure 38. Point to Point at 20 m/s with 10 m/s Wind from the North

Overall, the results of the active waypoint modification using the SDK interface were pleasing. While the algorithm was not optimal nor completely robust, it definitely improves the ability to put a sensor on a target using a small waypoint guided UAV operating in wind.

4.4 –Flight Testing with Wind Correction

Due to extenuating circumstances, the test team was unable to conduct the actual flight tests at the Area B test range. The tests were expected to be accomplished and were thoroughly planned. Official test cards, provided in Appendix D, were produced and reviewed. Unfortunately, the actual flight tests had to be postponed past the date of the thesis defense. Therefore, it is recommended that before any future lab testing is undertaken, a series of flight tests be conducted to validate the results obtained using the wind correction in the HITL simulation.

4.4.1 – Real Time Wind Estimating

- See Appendix E -

4.4.2 – Turn Rate & Updating "Rabbit" Waypoint Approaches

- Flight Test Postponed -

4.4.3 – Wind Corrected Sensor Pointing

- Flight Test Postponed -

4.5 – Chapter Conclusions

Chapter IV presented the results of the SIG Rascal UAV flight tests performed in the HITL simulation under the control of the standard Piccolo II autopilot as well as with the wind correction implemented. A set of baseline flight tests were conducted to determine the standard characteristics of the simulated aircraft flying in a windy environment. The findings revealed that the track following characteristics of the Piccolo II were quite good under the presence of a wind, and that the relative importance of this trait could be easily adjusted through the track convergence gain. The level of precision flight illustrated by the autopilot actually led to the primary focus and contribution of this thesis, the method of wind correction for sensor pointing. The crab angle induced by the controller to keep the aircraft on track resulted in a fixed sensor, such as a video feed, to survey areas well off track. To counter this effect, a wind correction was developed and implemented in the SDK code, which successfully updated and placed new waypoints for the UAV to track. These new waypoints adjusted the aircraft's flight path enough to allow the sensor footprint to cover the designated target. The wind correction worked well for straight line tracks and for more intricate tracks when flying at lower speeds. However, with higher speeds the simulated aircraft could not successfully adjust for the wind and sensor pointing.

One additional point must be factored in. The plots of the sensor footprint only represented the exact center of that footprint. In actuality, the footprint was hundreds of meters in diameter due to the field of view. Thus, when the center of the sensor crosses within 20, or even 50, meters, this was a desirable result. The sensor would then easily be able to supply adequate coverage of the targets.

V. Conclusions and Recommendations

5.1 – Conclusions

The research accomplished in this thesis project provids a solid foundation for future evaluation of small UAVs flying under the influence of winds. Initial baseline tests were performed to discern the standard capabilities of the COTS Piccolo II autopilot in conjunction with the SIG Rascal 110 aircraft. The physical component setup offered realistic measurements and data which could easily be applied to an operational environment. This investigation into wind compensation methods achieved several key objectives:

1) Collected a baseline set of data which represents the wind compensation capabilities of a COTS autopilot implemented in a true life setting.

2) Formulation and implementation of a real time update of the current wind direction and velocity that the aircraft was encountering. Using the SDK interface, the operator can now view and log the real time winds along the UAVs true flight path. The output data were not completely without some outliers, but the overall result was acceptable.

3) Formulation of three differing approaches for employing wind corrections for a UAV. The first utilized a direct implementation of a new aircraft heading and airspeed required, based on the wind estimation described above. The second method employed a continuously updating unattainable “rabbit” waypoint that would mislead the aircraft into reaching the desired original waypoint. The third technique took a completely different approach to wind correction and adjusted the aircraft’s flight path based on the position of a sensor footprint rather than the position of the UAV. It was determined that despite

an accurate flight path along the determined track, the nose of the UAV would not necessarily be pointed straight ahead. For this reason, the sensor may not survey the desired target and overall mission effectiveness would not be satisfied without a real time modification to the original flight plan.

4) Demonstrated the interfacing ability of the SDK software to receive, process, and then transmit new flight parameters to the on-board autopilot unit. Real time aircraft telemetry, waypoint data, and track commands were all communicated to and from the UAV using the C++ program developed with the SDK.

The initial research plan focused on improvement in the precise track following capabilities of small UAVs. The most challenging flight conditions were reconciled as a precise track following mission that would be required in the "urban canyon" environment. While recognizing that operationally deployed autonomous small UAVs navigating amongst buildings, trees, etc. is a few years in the future, the research presented on the turn rate and "rabbit" wind correction approaches provides a good basis from which future studies should be conducted. However, the crux of this thesis morphed into the active flight path modifications for precise sensor pointing. Research showed that this topic has not been previously addressed, yet is more applicable to current operational tasks for small UAVs than those mentioned above. Thus, while it was important to provide a solid framework for the more conventional methods of wind corrections, the sensor pointing problem was more pertinent and became the central focus.

The overall results of the new research focus were promising as the UAV tracked the predetermined flight paths very well under reasonable TAS and wind conditions.

However, at the higher speeds (>TAS=30 m/s, or with wind components of more than 50% of the TAS) the aircraft could not navigate accurately. These are considered extreme cases in an operational environment. In the normal flight regimes the incorporated wind corrections proved generally acceptable. More specifically, the sensor pointing approach showed that an algorithm could be implemented which would appreciably remove or reduce the sensor pointing errors. Undoubtedly, with subsequent research, this algorithm could be developed to be extremely robust and effective across the small UAV operational environment.

5.2 – Recommendations

The following recommendations incorporate improvements to the algorithms, interfacing procedure, and flight testing program along with follow on research guidance and suggestions.

- Incorporate actual flight tests to support the simulated data. Actual tests were planned, but did not happen due to operating restrictions beyond the control of the research team. This data will be particularly pertinent because indications from previous research (Jodeh, 2006) suggests there may be differences between simulated and actual flight performance.
- "Hard coding" information reduced the robustness of the current code. "Soft code" as much information as possible into any computer program.
- Acquire a larger volume of test and evaluation airspace. The bounds set out for the Area B test flight airspace was quite restrictive in both length

and altitude. In order to fully test and evaluate these UAVs a much larger space is recommended.

- Follow-on research should include: Implementation of both the turn rate and “rabbit” approaches, solidifying the sensor pointing method, and integration of related multiple research topics (e.g. formation flight, situational awareness, etc.) using the Piccolo SDK.

Appendix A: Complete Set of Simulated Test Results

Baseline Tests

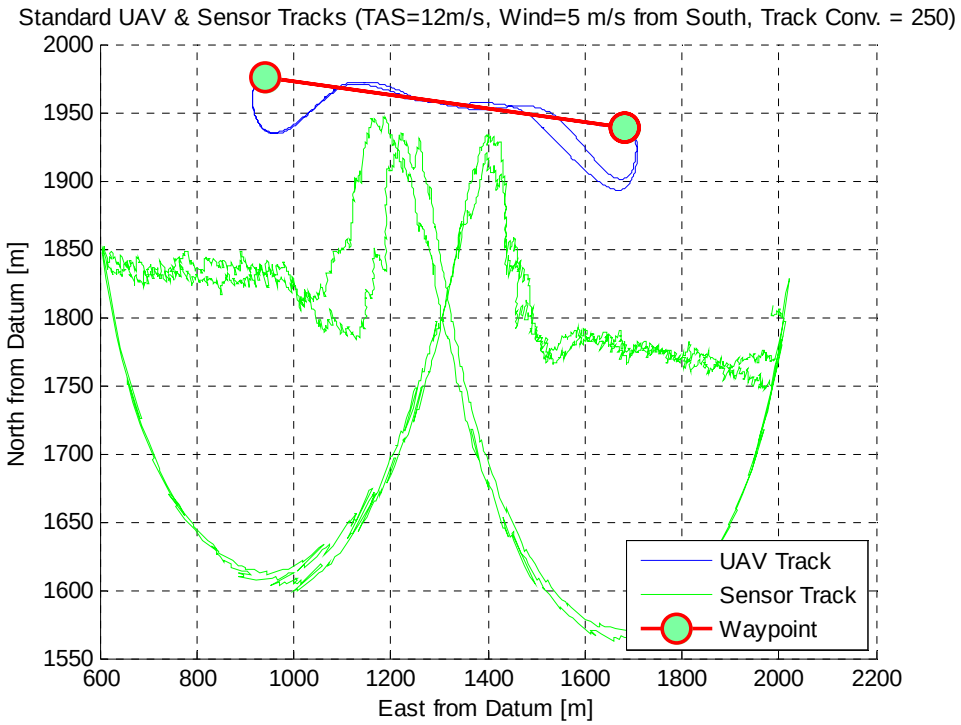

Figure 39. Standard UAV Short Point to Point at 12 m/s with Wind=5 m/s

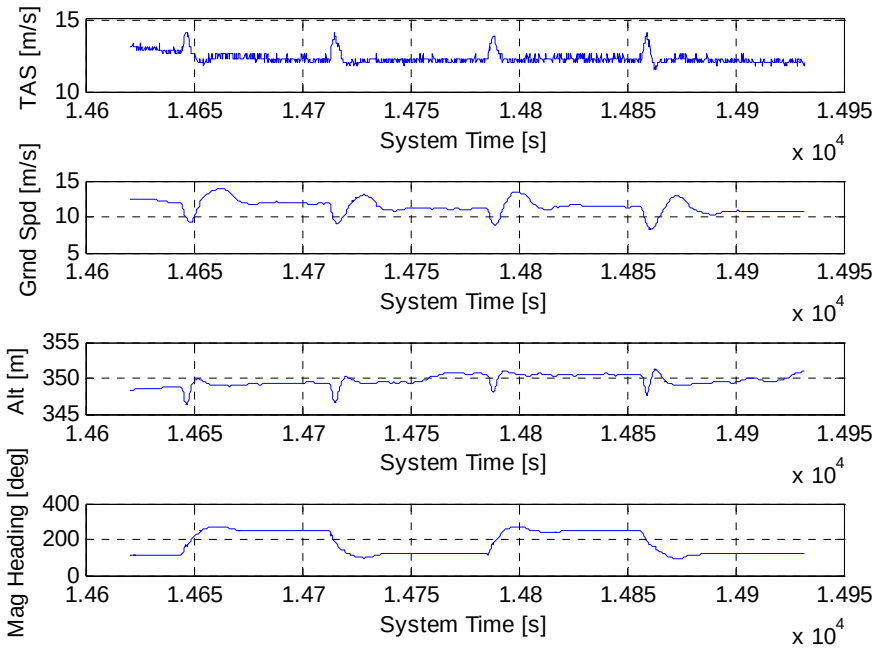

Figure 40. Various Parameters for Short Point to Point at 12 m/s

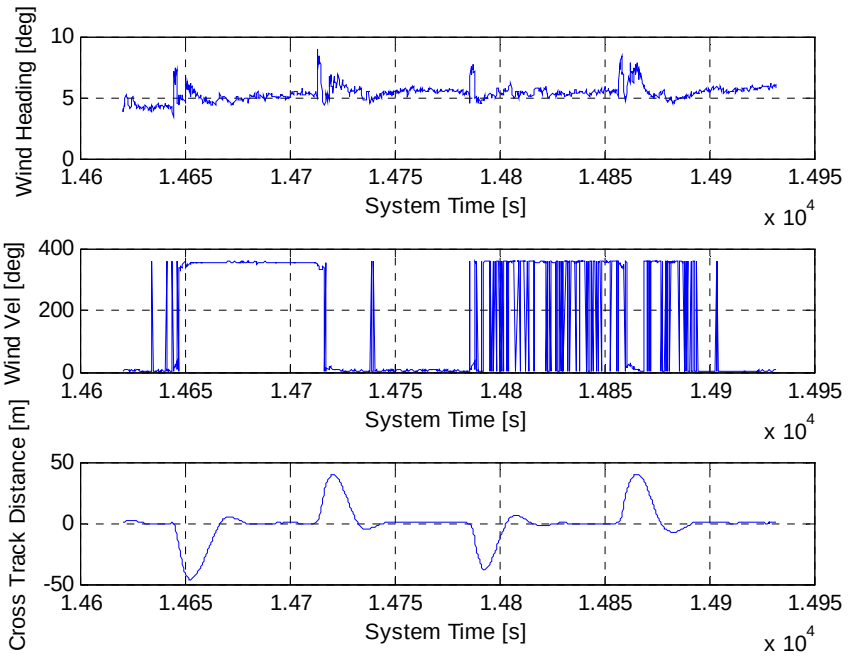

Figure 41. Real Time Wind Estimations for Short Point to Point at 12 m/s

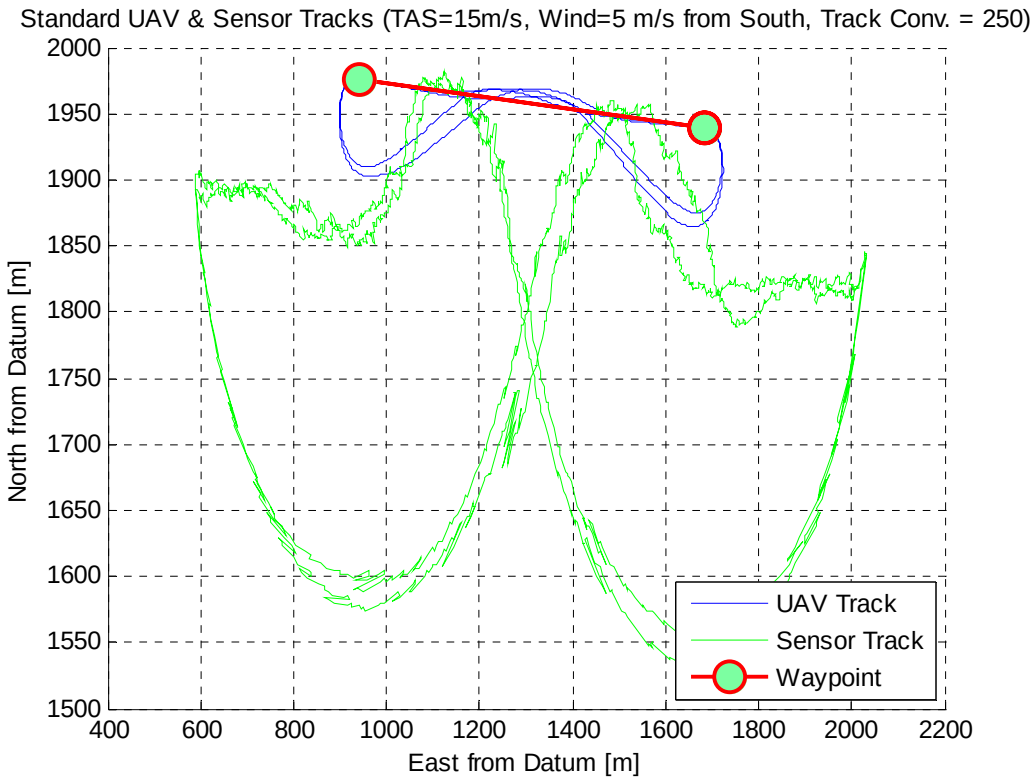

Figure 42. Standard UAV Short Point to Point at 15 m/s with Wind=5 m/s

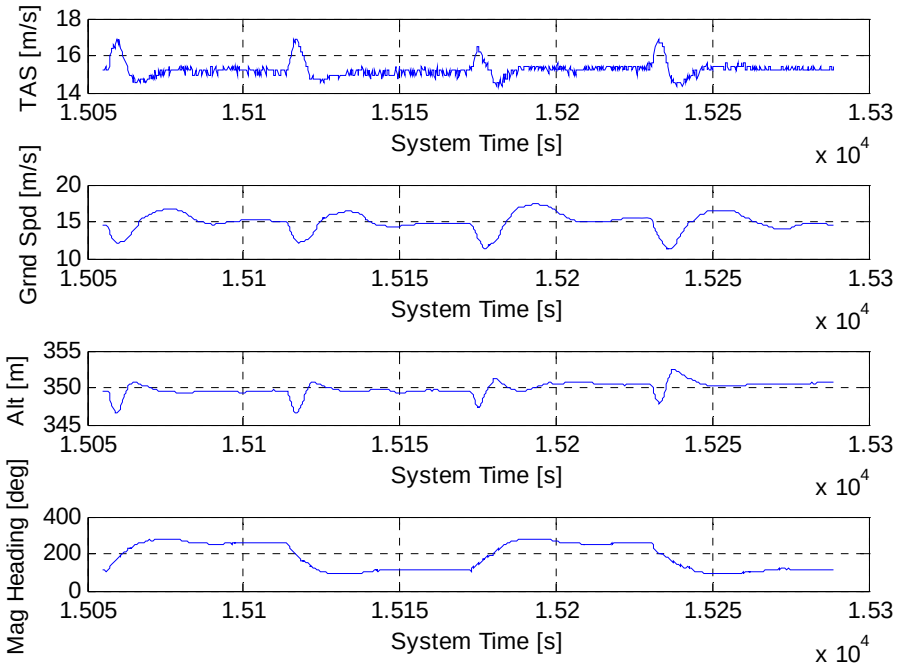

Figure 43. Various Parameters for Short Point to Point at 15 m/s

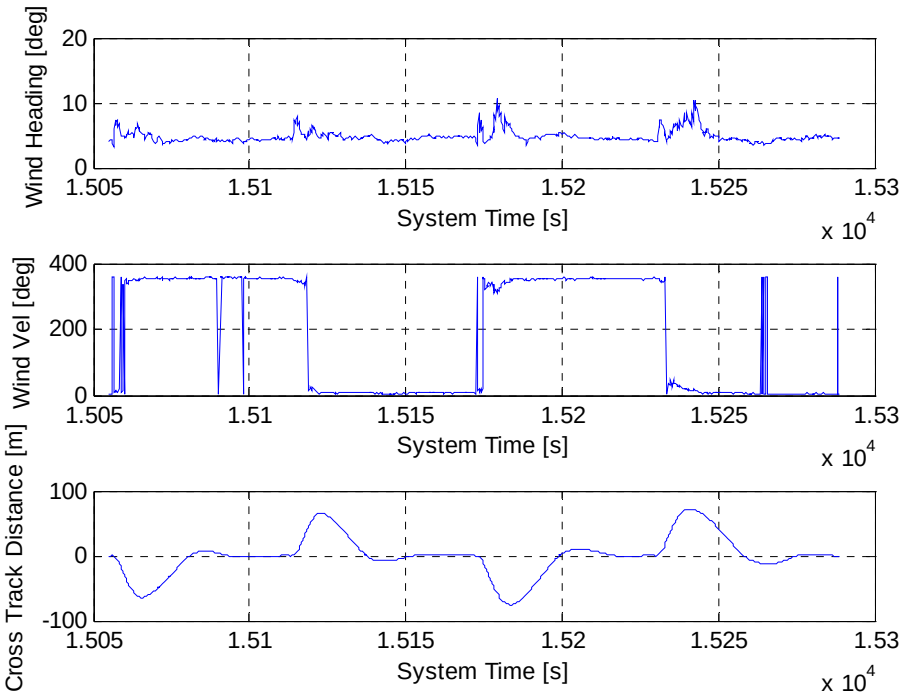

Figure 44. Real Time Wind Estimations for Short Point to Point at 15 m/s

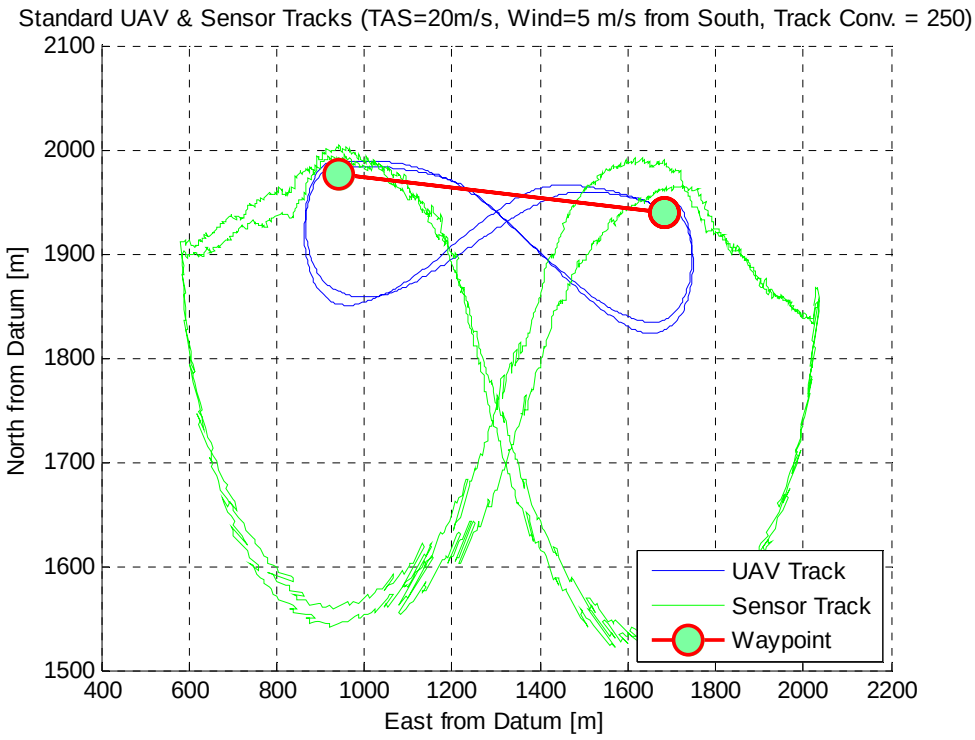

Figure 45. Standard UAV Short Point to Point at 20 m/s with Wind=5 m/s

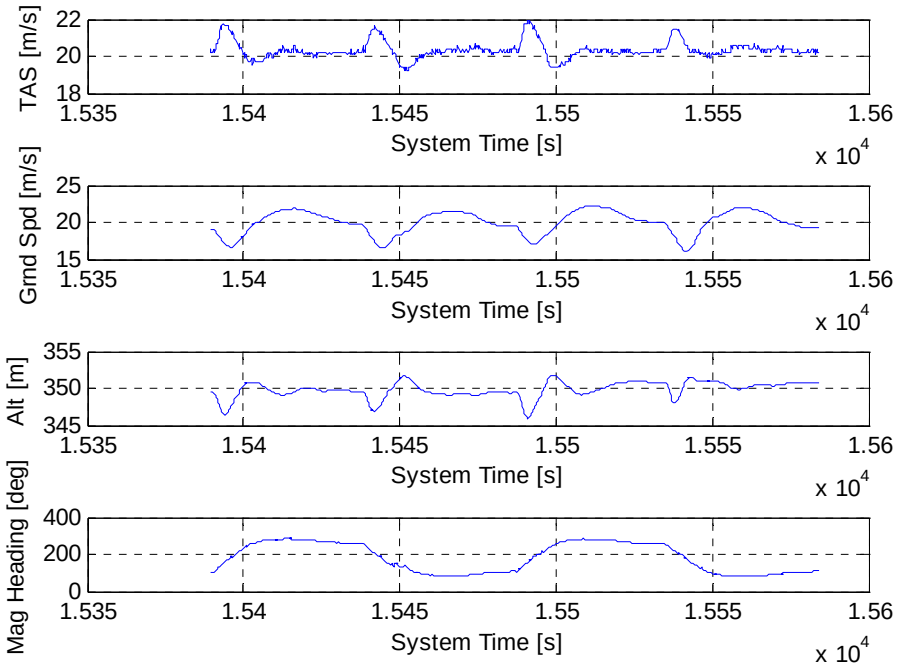

Figure 46. Various Parameters for Short Point to Point at 20 m/s

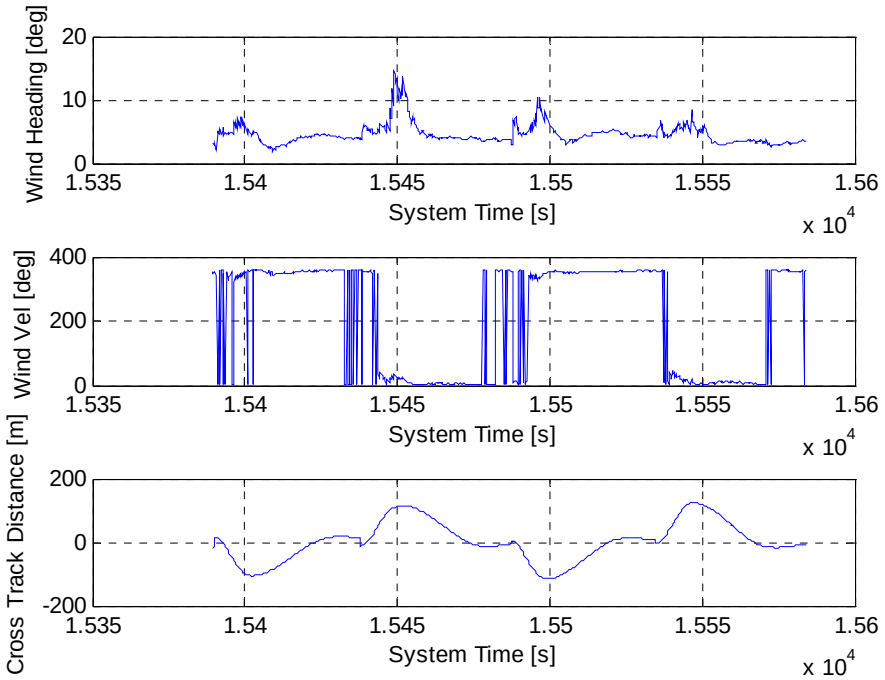

Figure 47. Real Time Wind Estimations for Short Point to Point at 20 m/s

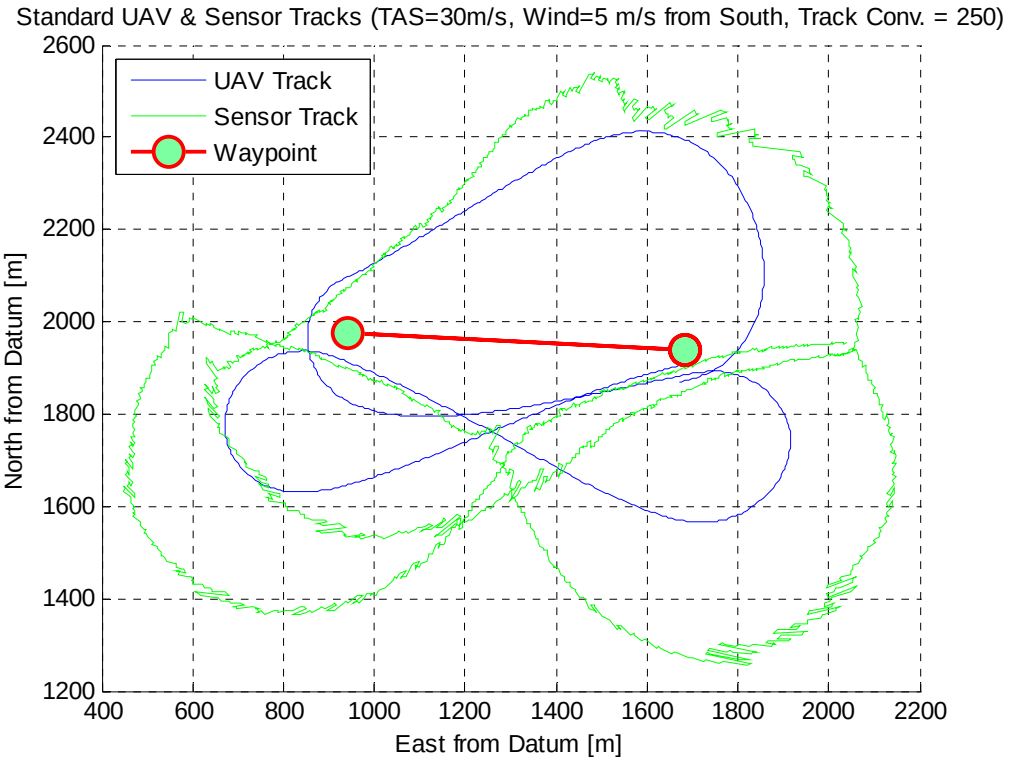

Figure 48. Standard UAV Short Point to Point at 30 m/s with Wind=5 m/s

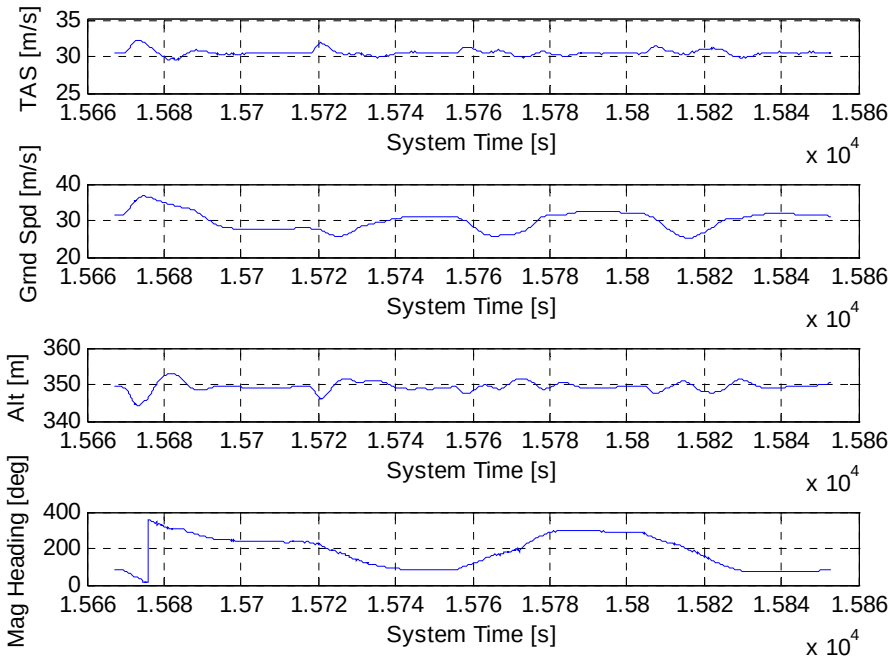

Figure 49. Various Parameters for Short Point to Point at 30 m/s

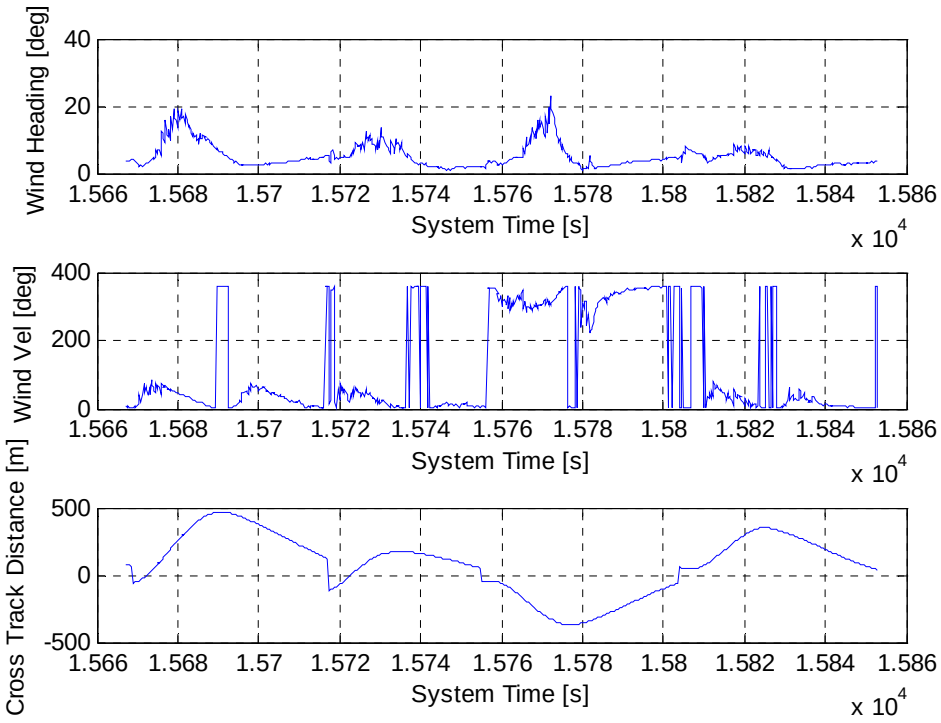

Figure 50. Real Time Wind Estimations for Short Point to Point at 30 m/s

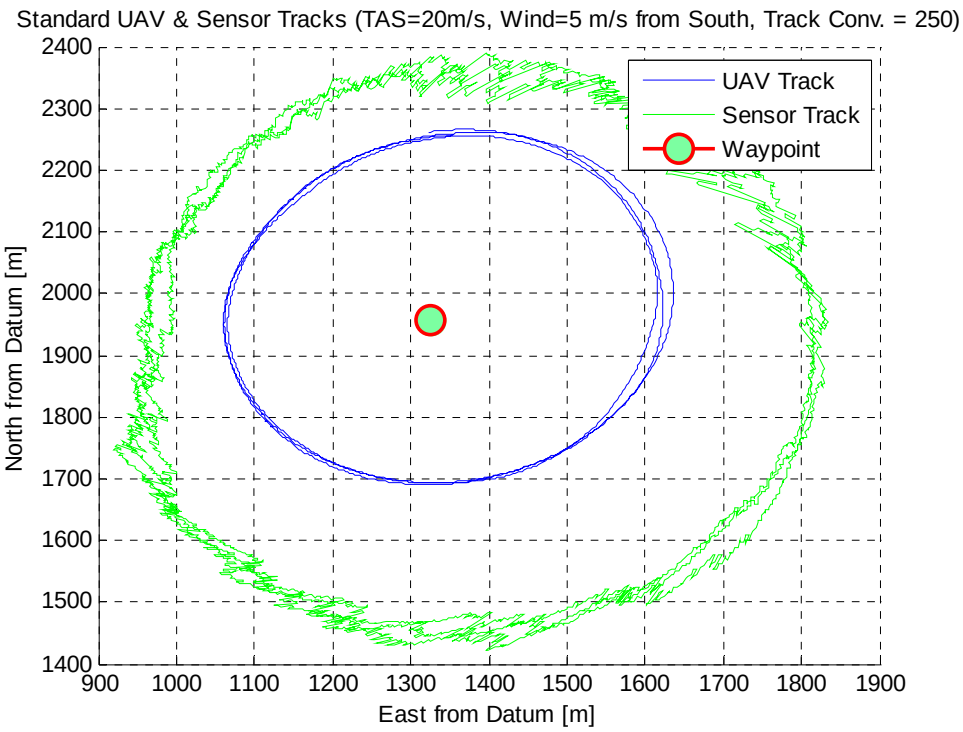

Figure 51. Standard UAV Circular Orbit at 20 m/s

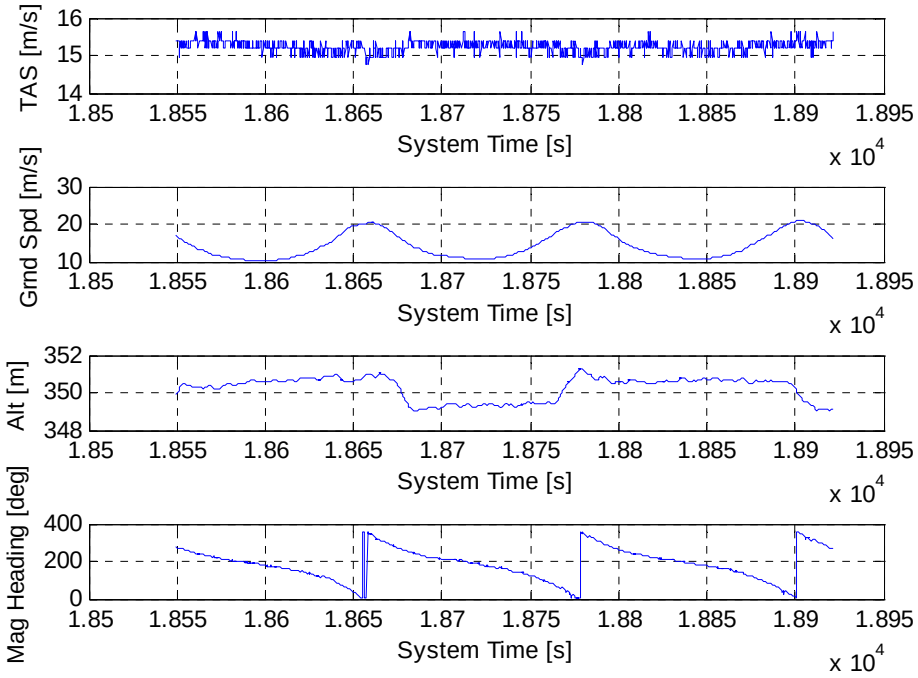

Figure 52. Various Parameters for the Circular Orbit at 20 m/s

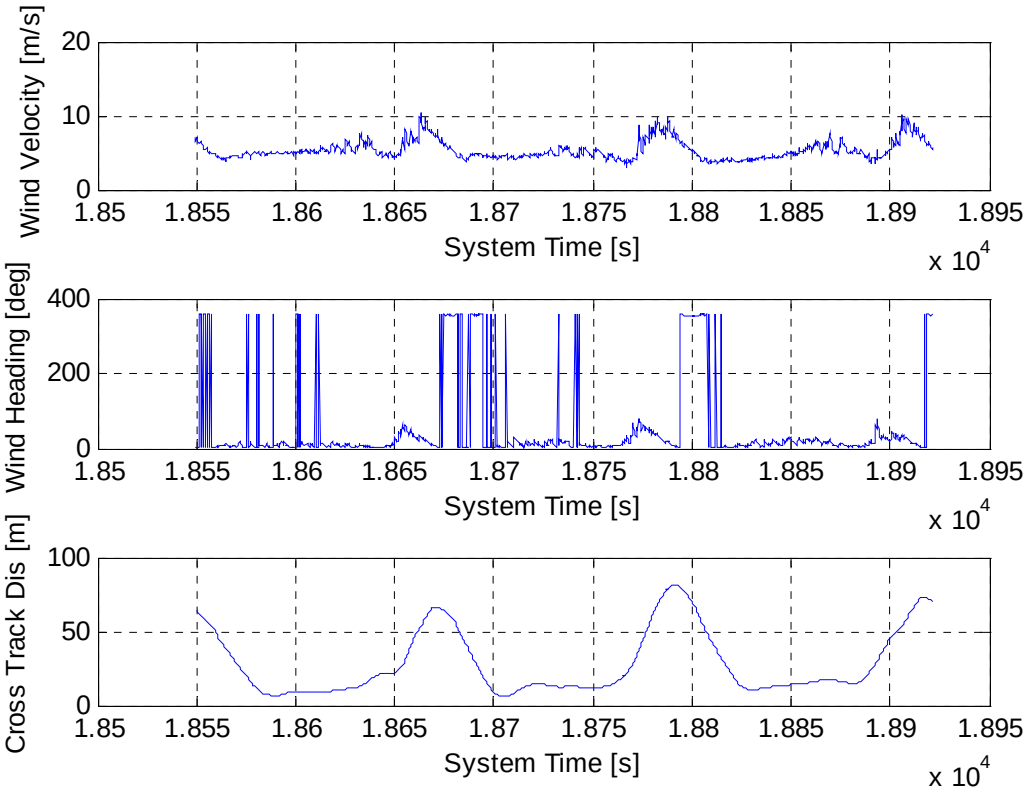

Figure 53. Real Time Wind Estimations for the Circular Orbit at 20 m/s

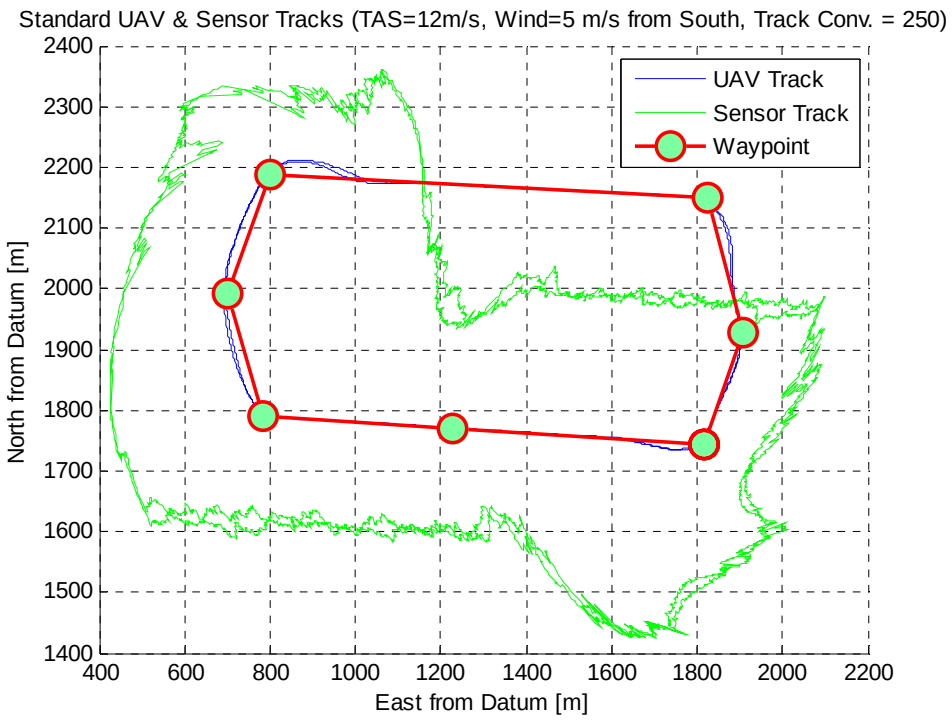

Figure 54. Standard UAV Race Track Pattern at 12 m/s with Wind=5 m/s and TC=250

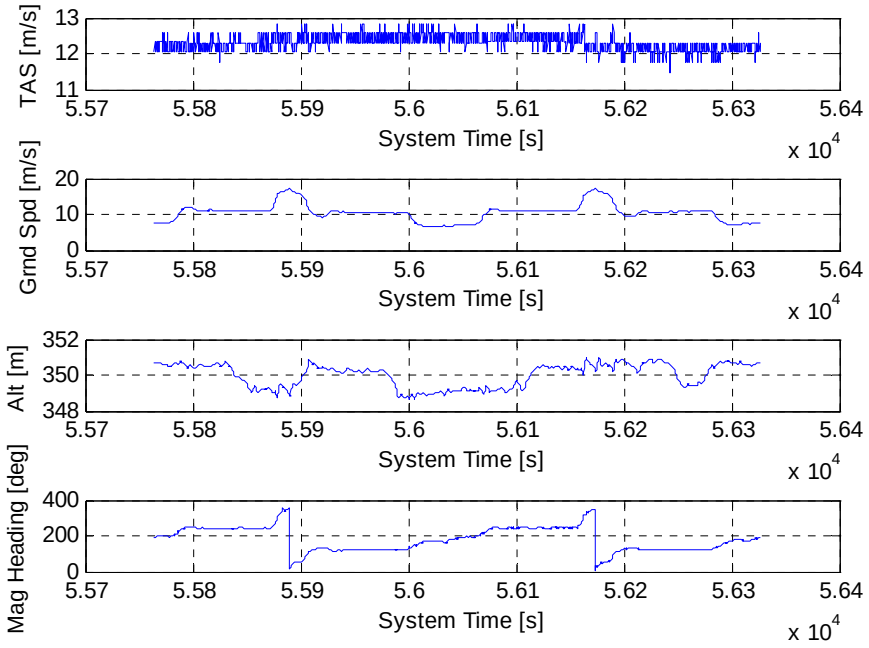

Figure 55. Various Parameters for the Race Track Pattern at 12 m/s, Wind5 m/s, & TC=250

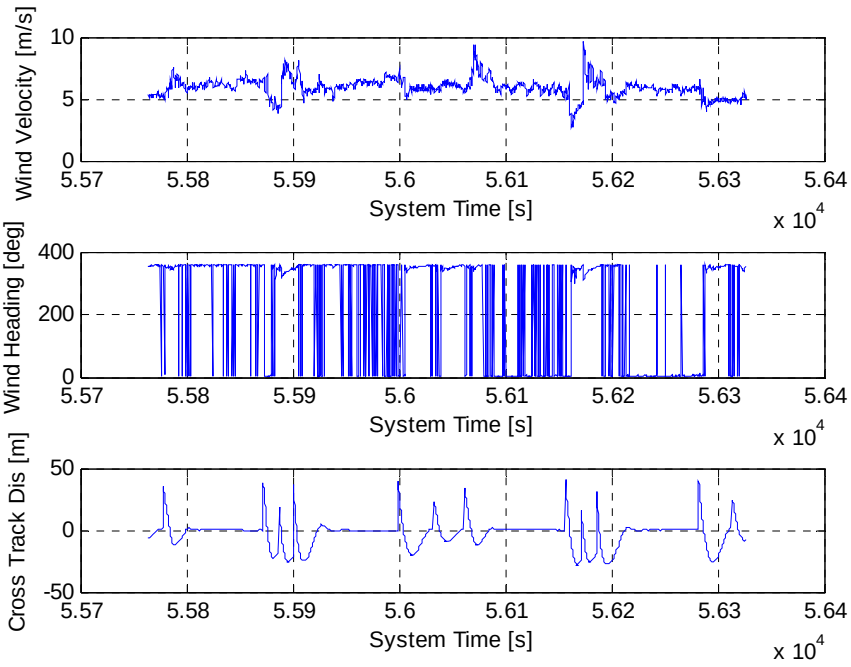

Figure 56. Real Time Wind Estimations for the Race Track at 12 m/s, Wind=5 m/s, & TC=250

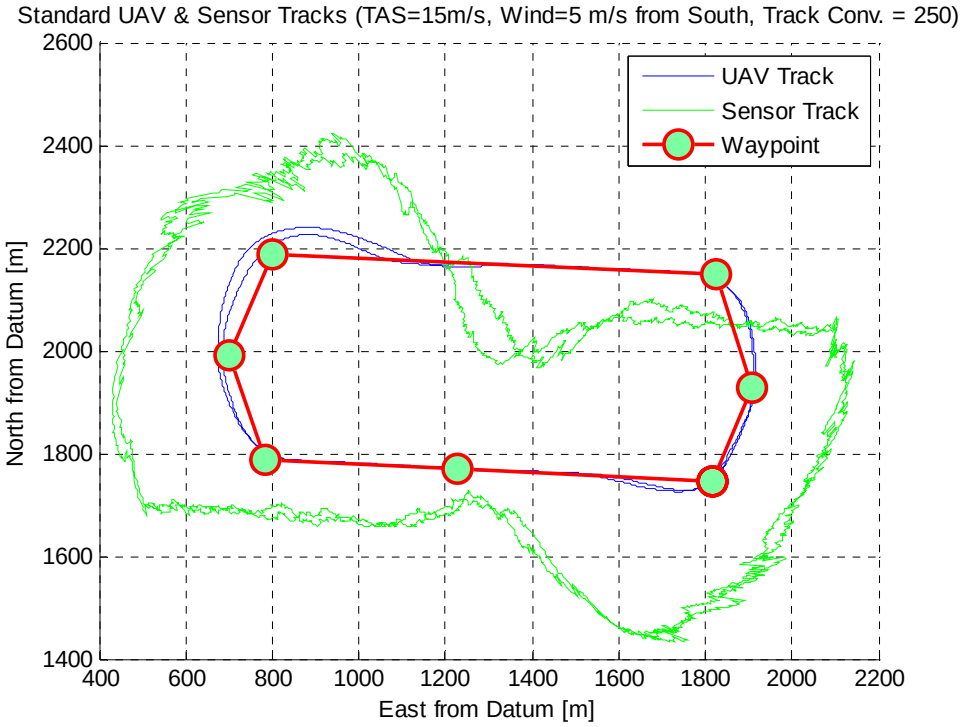

Figure 57. Standard UAV Race Track Pattern at 15 m/s with Wind=5 m/s and TC=250

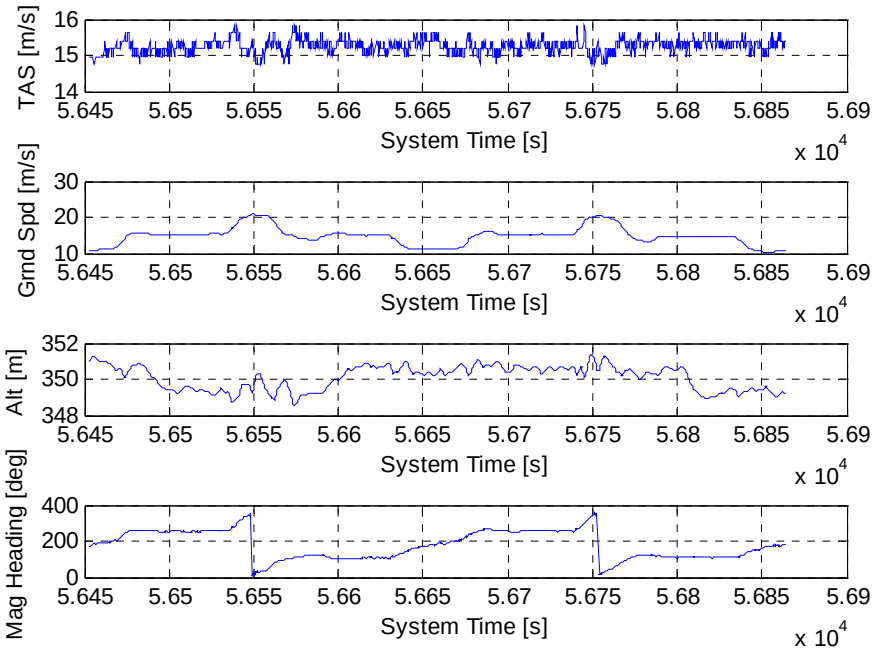

Figure 58. Various Parameters for the Race Track Pattern at 15 m/s, Wind5 m/s, & TC=250

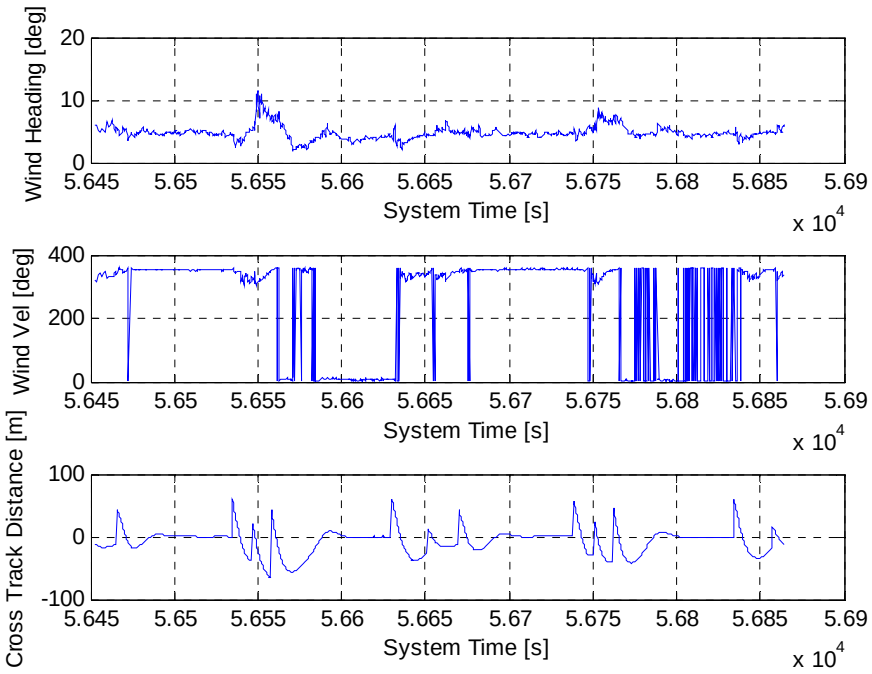

Figure 59. Real Time Wind Estimations for the Race Track at 15 m/s, Wind=5 m/s, & TC=250

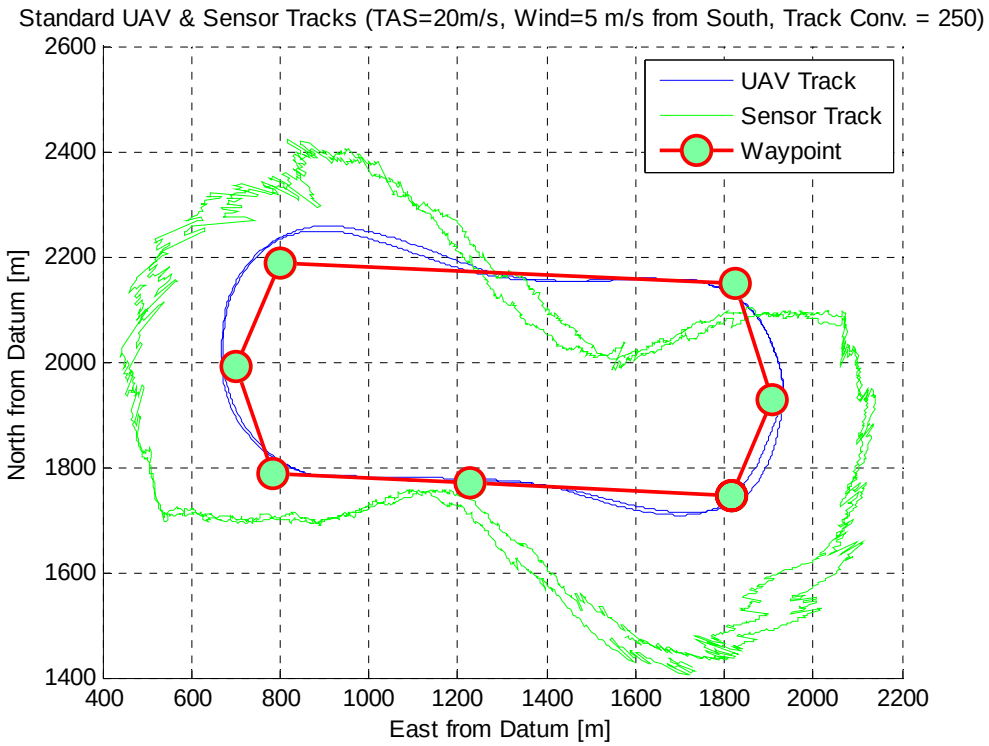

Figure 60. Standard UAV Race Track Pattern at 20 m/s with Wind=5 m/s and TC=250

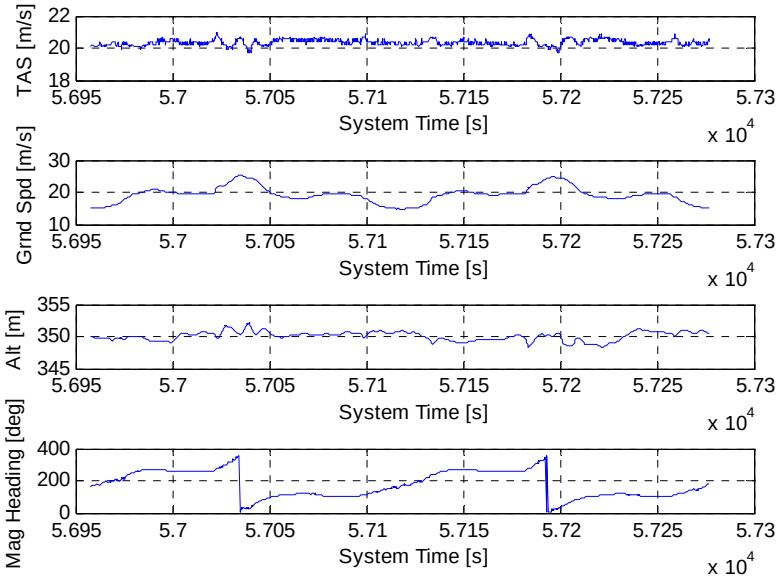

Figure 61. Various Parameters for the Race Track Pattern at 20 m/s, Wind5 m/s, & TC=250

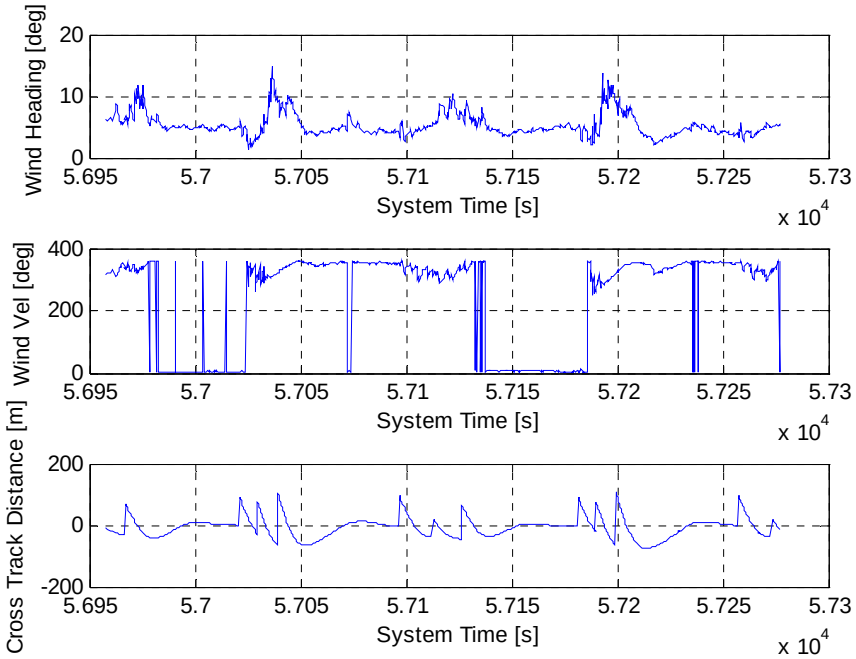

Figure 62. Real Time Wind Estimations for the Race Track at 20 m/s, Wind=5 m/s, & TC=250

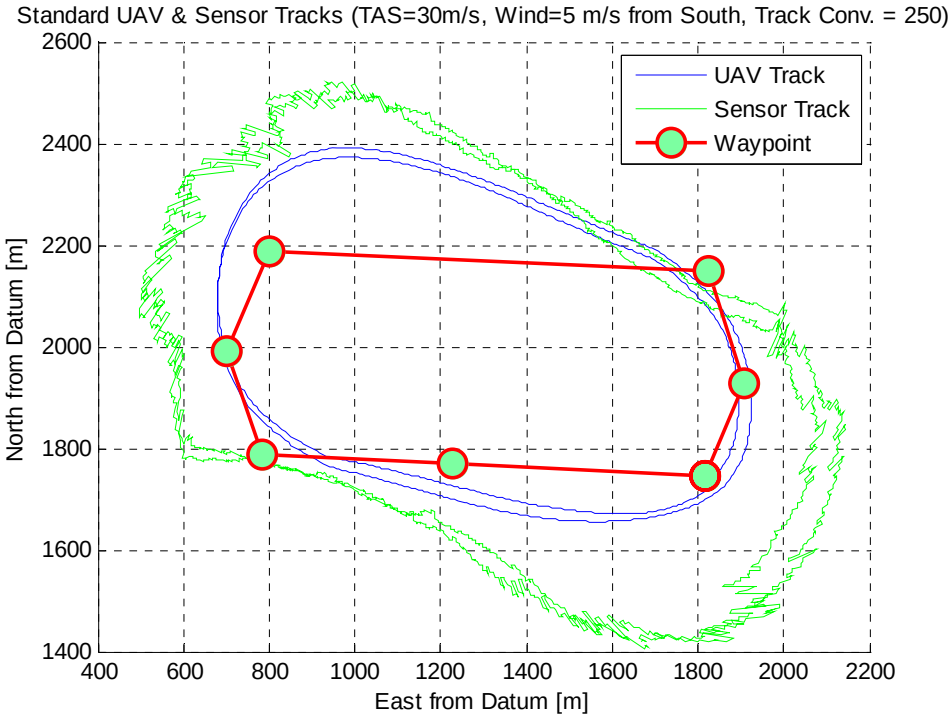

Figure 63. Standard UAV Race Track Pattern at 30 m/s with Wind=5 m/s and TC=250

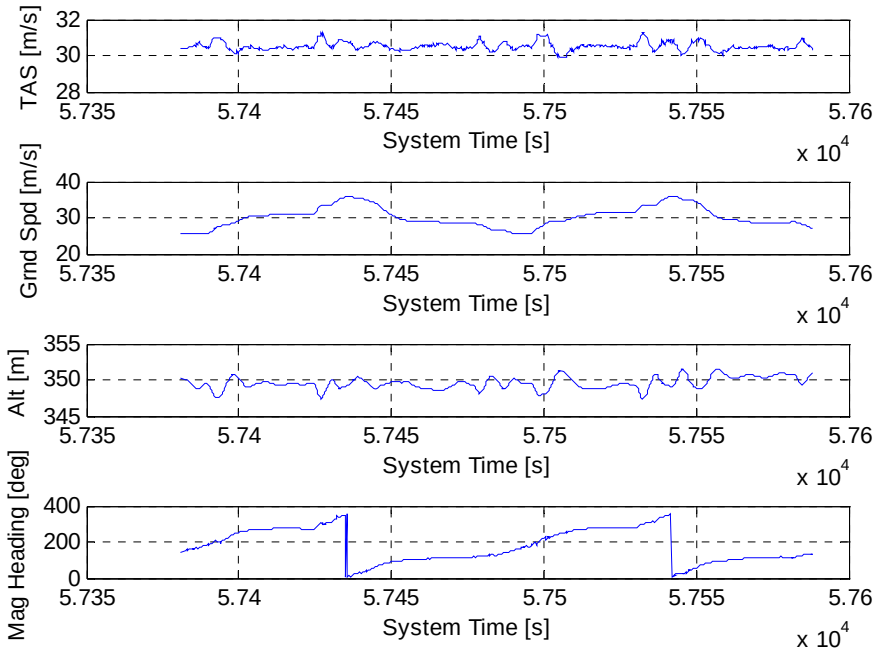

Figure 64. Various Parameters for the Race Track Pattern at 30 m/s, Wind5 m/s, & TC=250

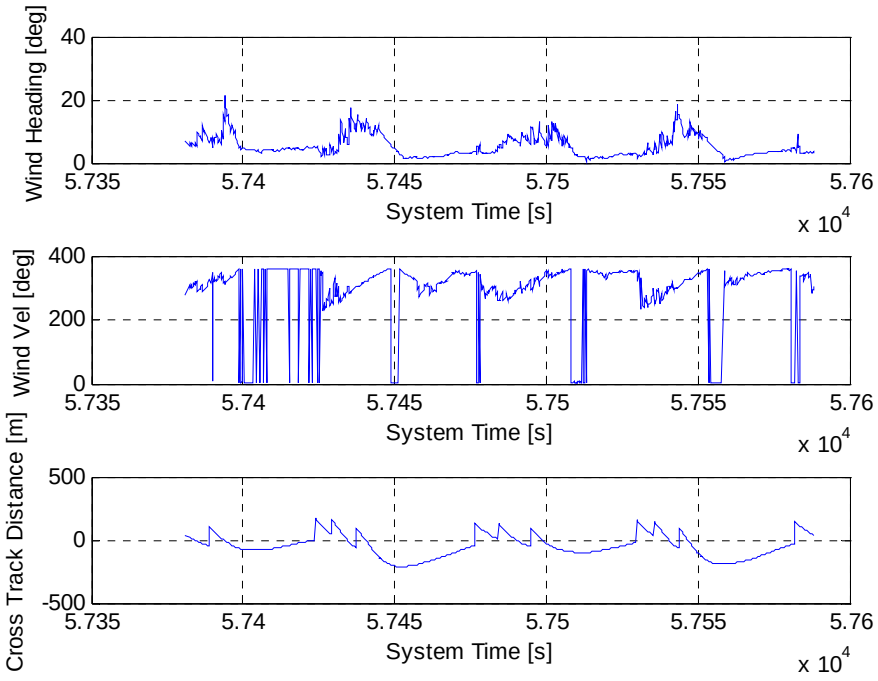

Figure 65. Real Time Wind Estimations for the Race Track at 30 m/s, Wind=5 m/s, & TC=250

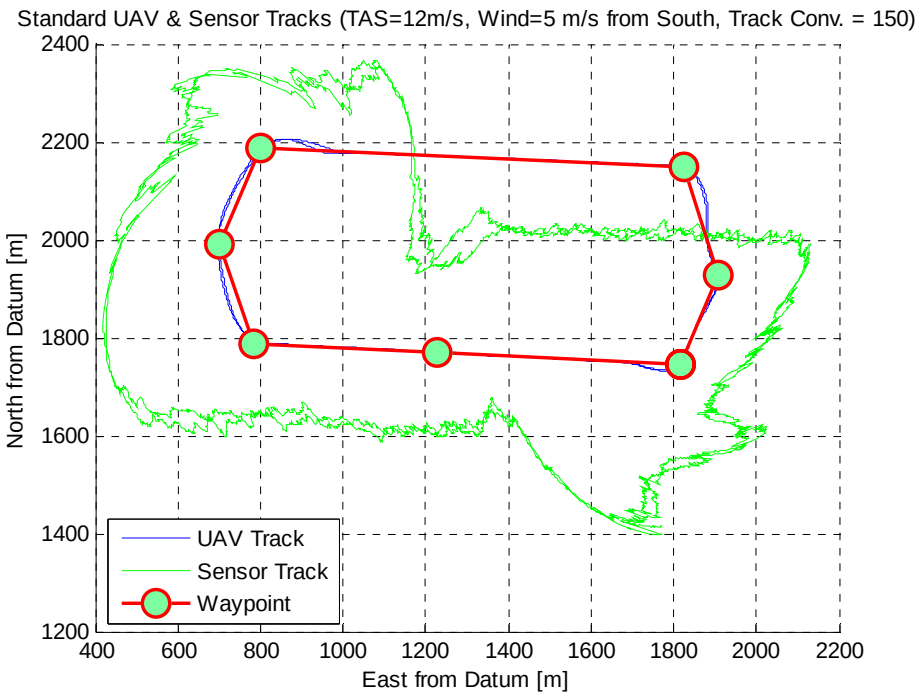

Figure 66. Standard UAV Race Track Pattern at 12 m/s with Wind=5 m/s and TC=150

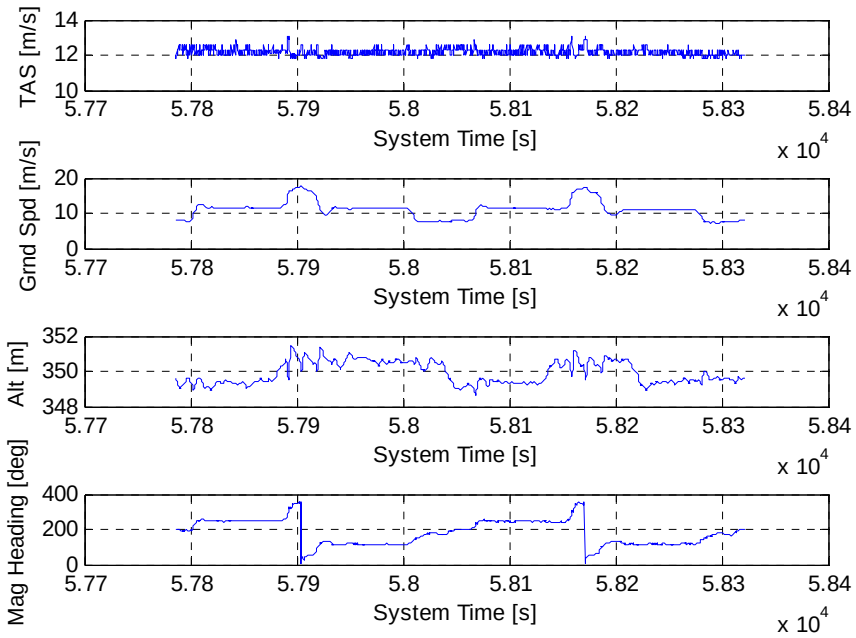

Figure 67. Various Parameters for the Race Track Pattern at 12 m/s, Wind5 m/s, & TC=150

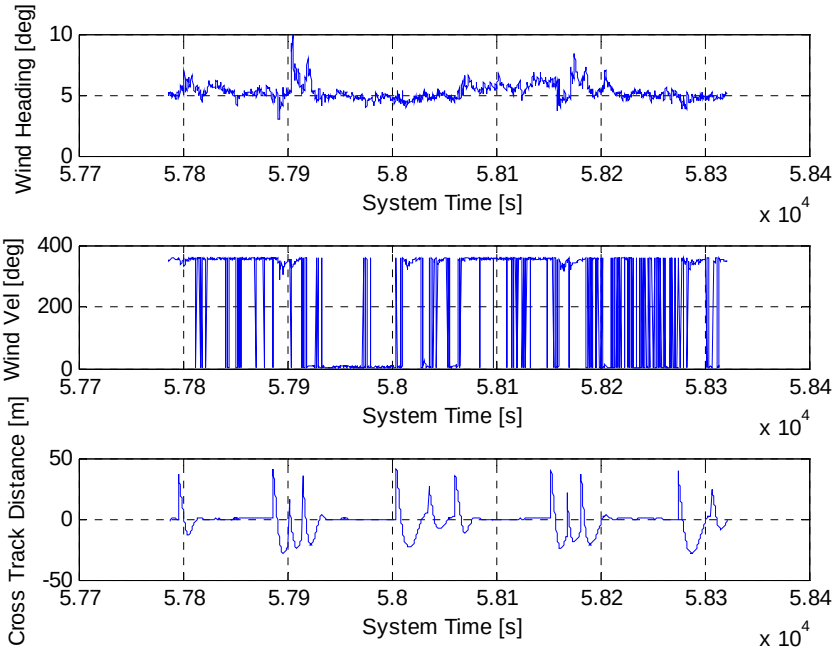

Figure 68. Real Time Wind Estimations for the Race Track at 12 m/s, Wind=5 m/s, & TC=150

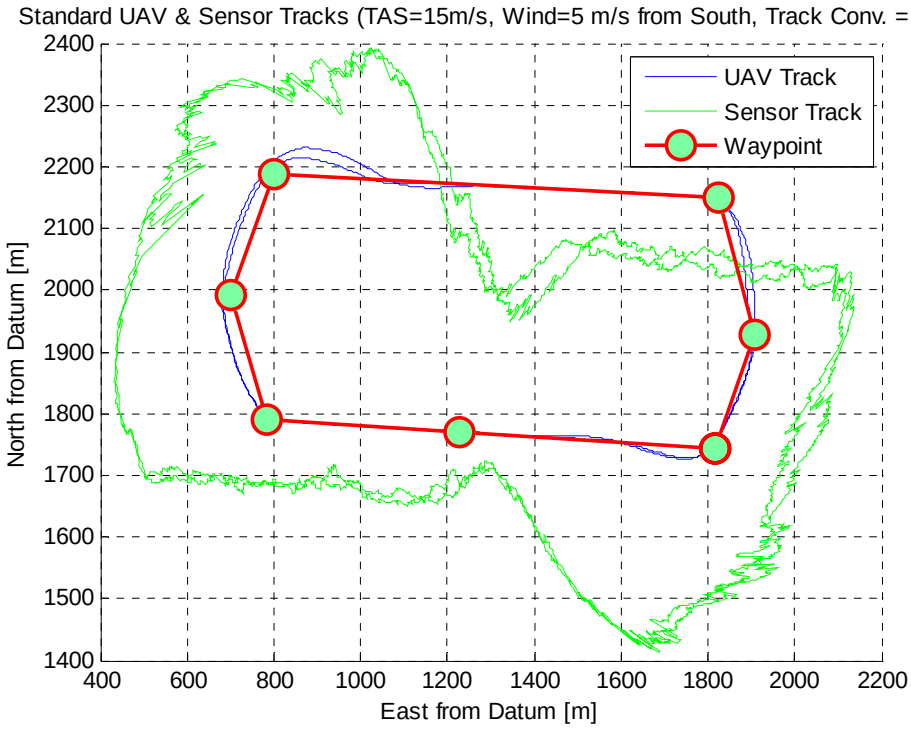

Figure 69. Standard UAV Race Track Pattern at 15 m/s with Wind=5 m/s and TC=150

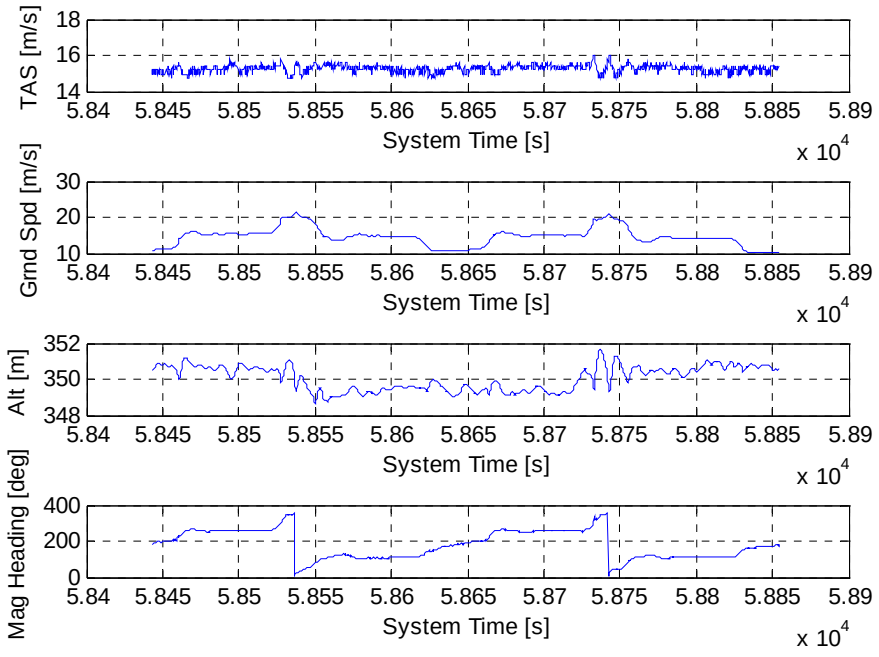

Figure 70. Various Parameters for the Race Track Pattern at 15 m/s, Wind5 m/s, & TC=150

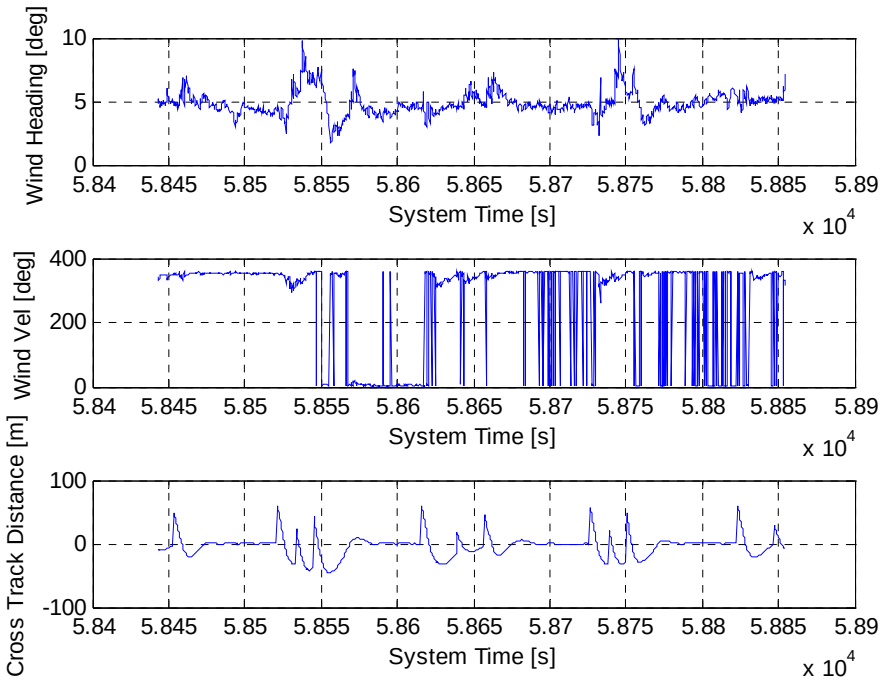

Figure 71. Real Time Wind Estimations for the Race Track at 15 m/s, Wind=5 m/s, & TC=150

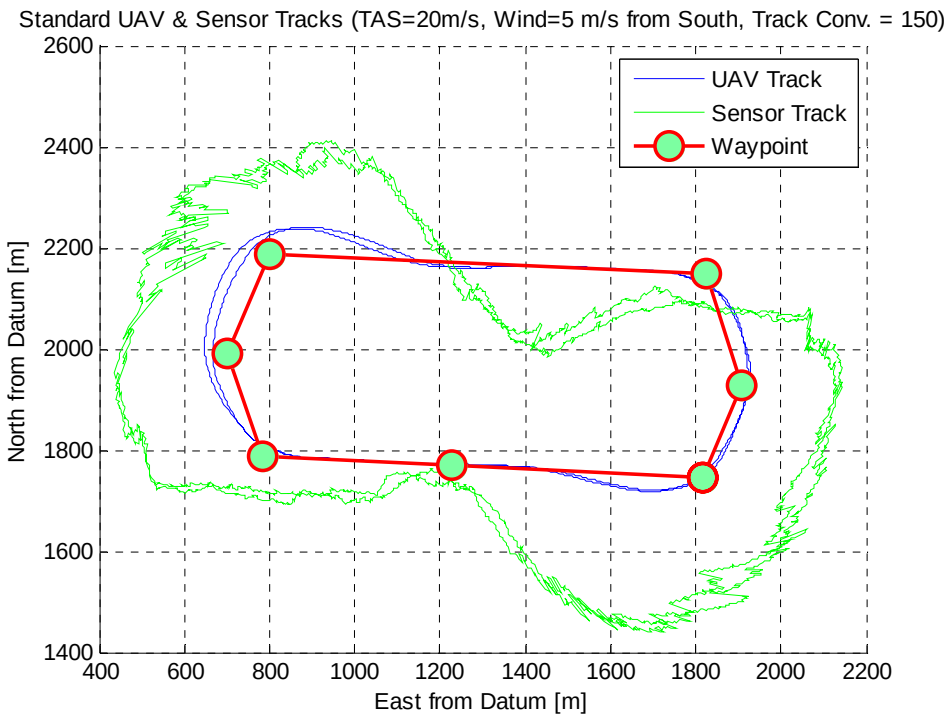

Figure 72. Standard UAV Race Track Pattern at 20 m/s with Wind=5 m/s and TC=150

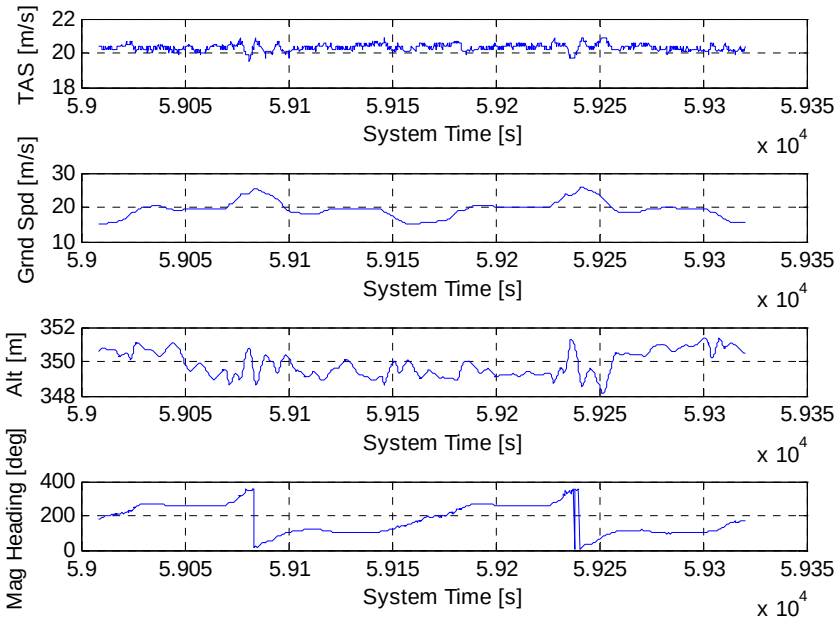

Figure 73. Various Parameters for the Race Track Pattern at 20 m/s, Wind5 m/s, & TC=150

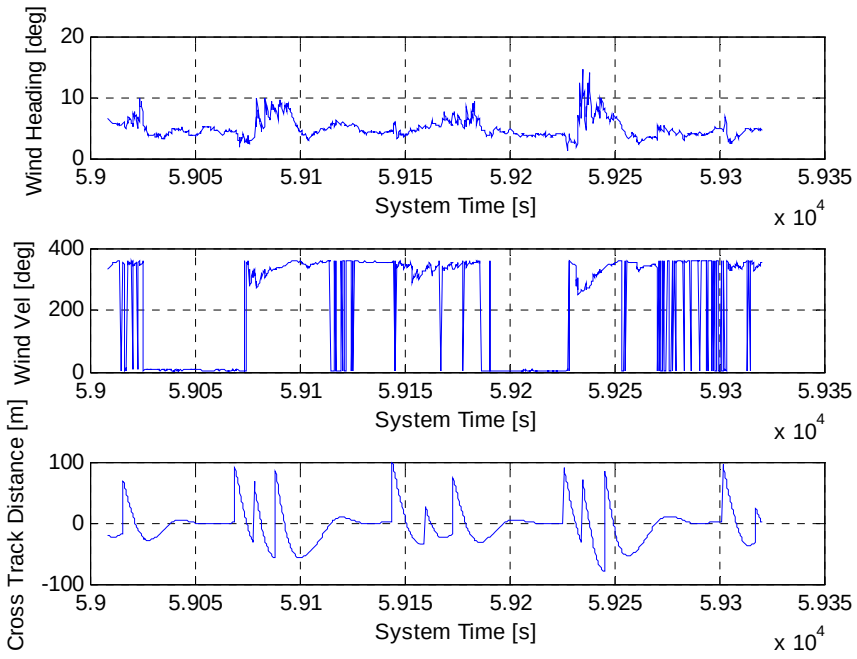

Figure 74. Real Time Wind Estimations for the Race Track at 20 m/s, Wind=5 m/s, & TC=150

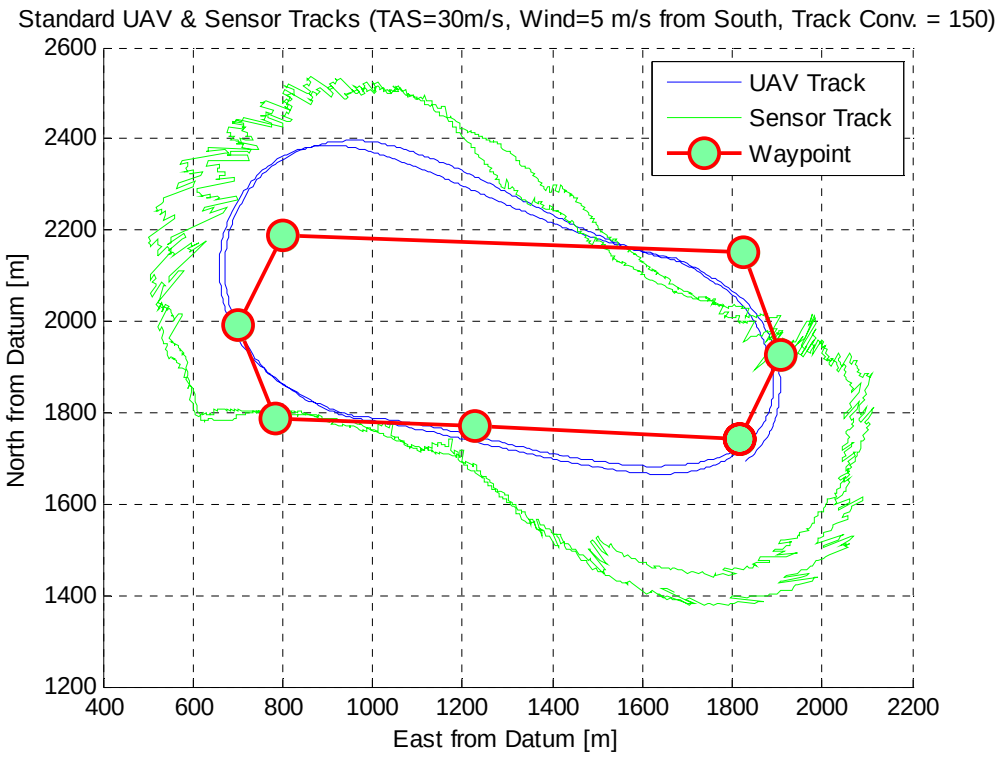

Figure 75. Standard UAV Race Track Pattern at 30 m/s with Wind=5 m/s and TC=150

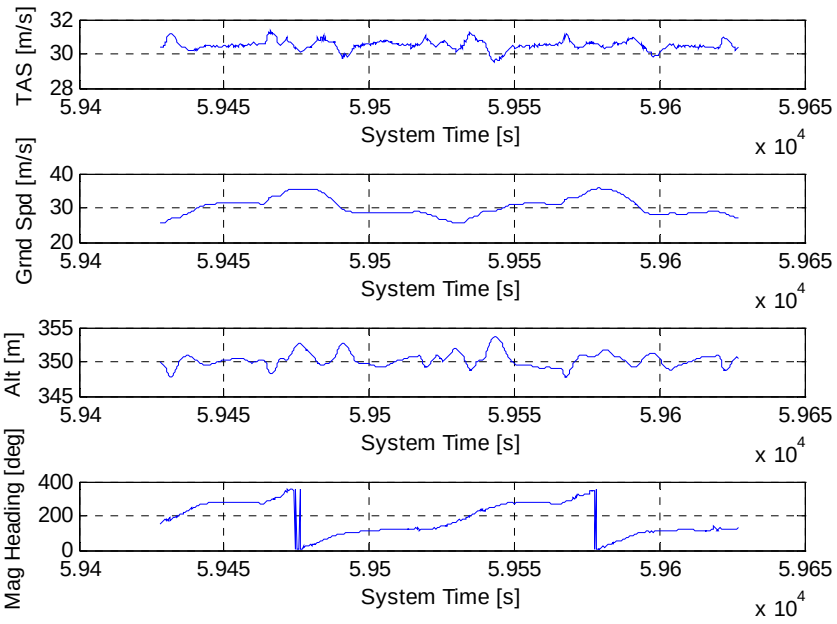

Figure 76. Various Parameters for the Race Track Pattern at 30 m/s, Wind5 m/s, & TC=150

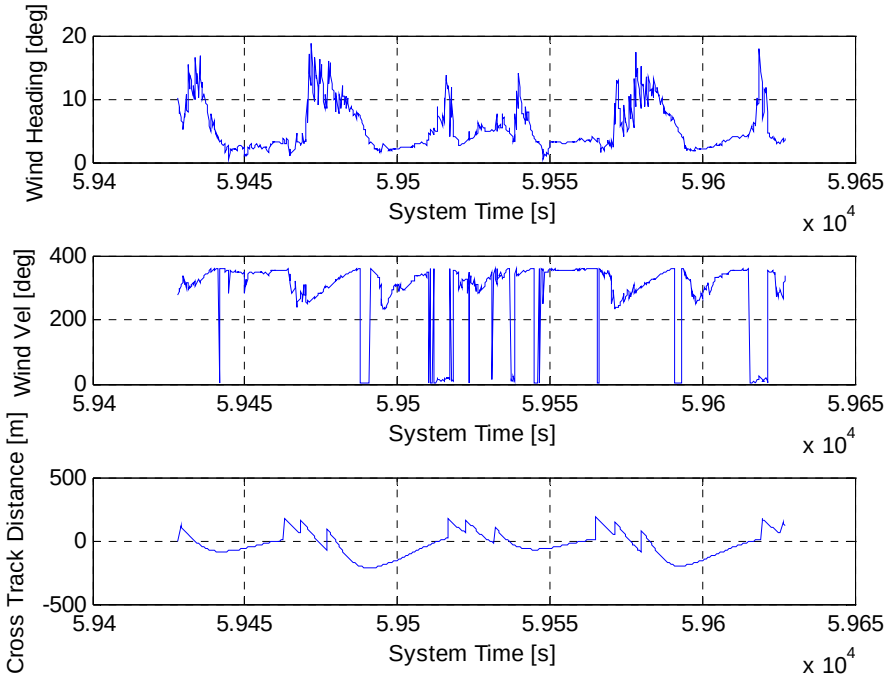

Figure 77. Real Time Wind Estimations for the Race Track at 30 m/s, Wind=5 m/s, & TC=150

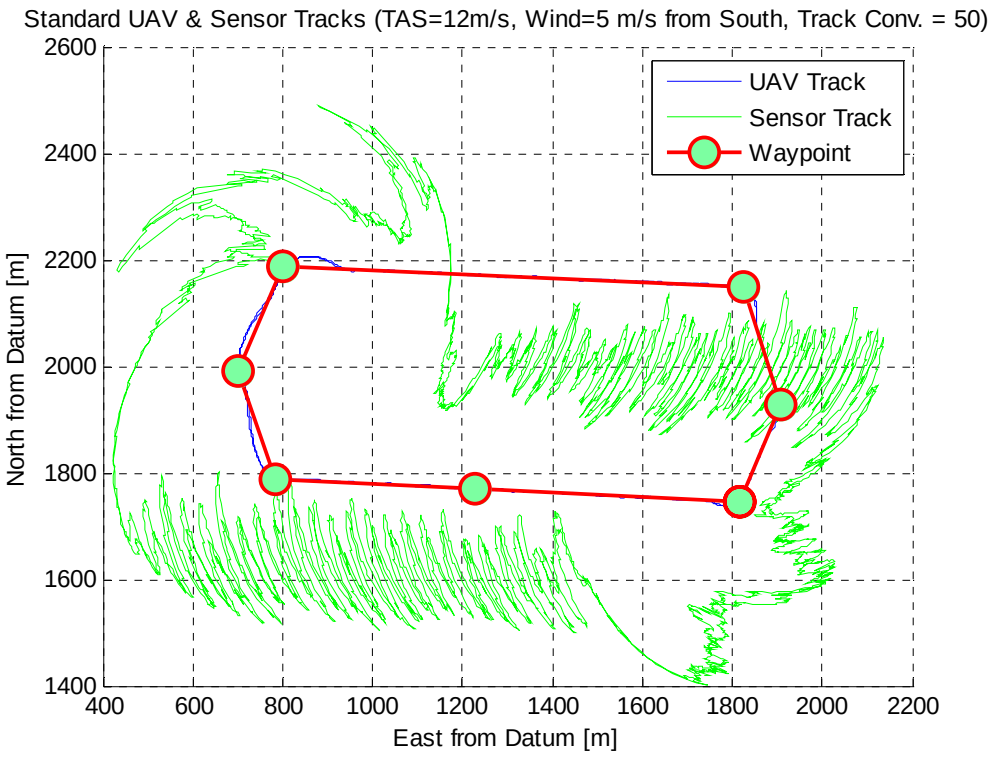

Figure 78. Standard UAV Race Track Pattern at 12 m/s with Wind=5 m/s and TC=50

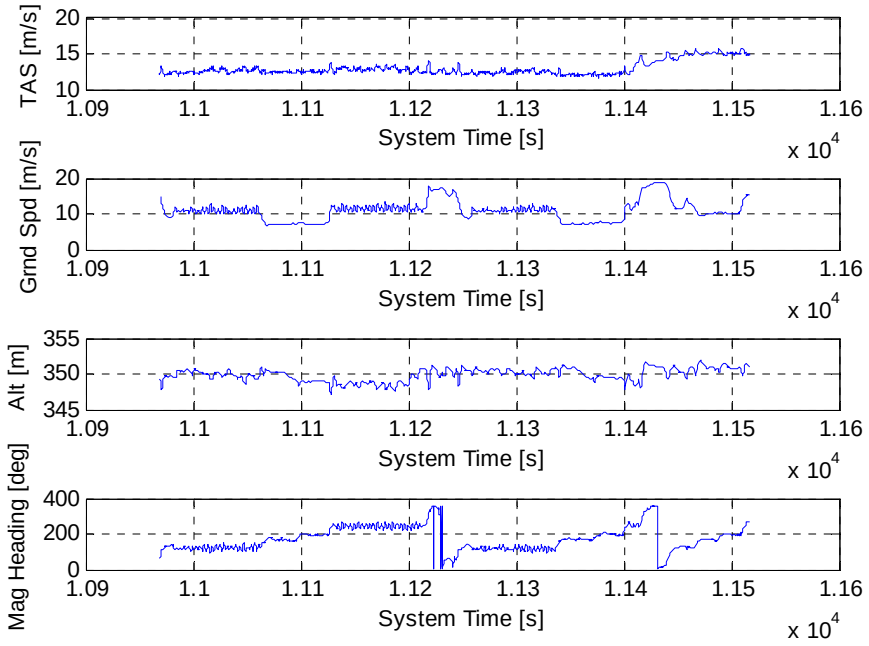

Figure 79. Various Parameters for the Race Track Pattern at 12 m/s, Wind5 m/s, & TC=50

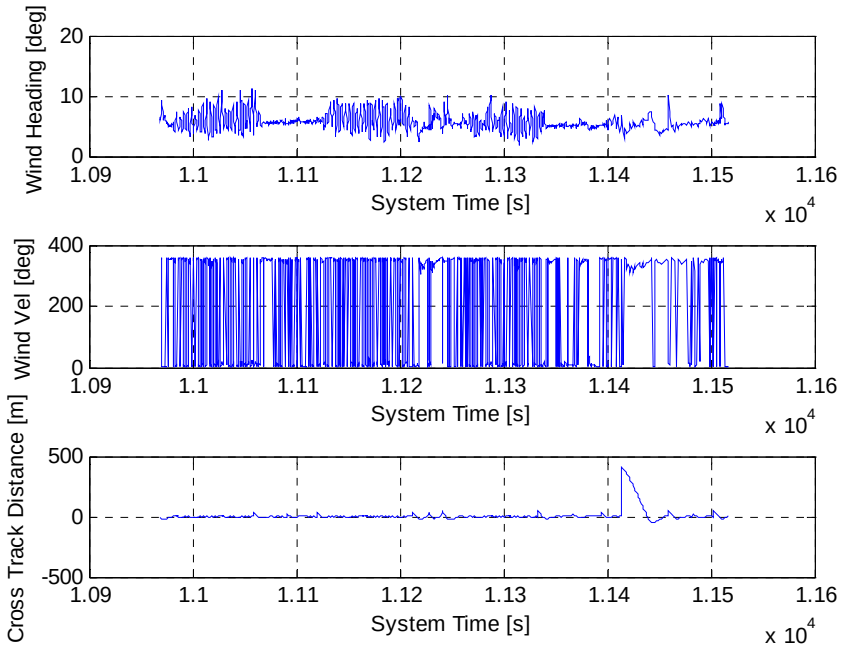

Figure 80. Real Time Wind Estimations for the Race Track at 12 m/s, Wind=5 m/s, & TC=50

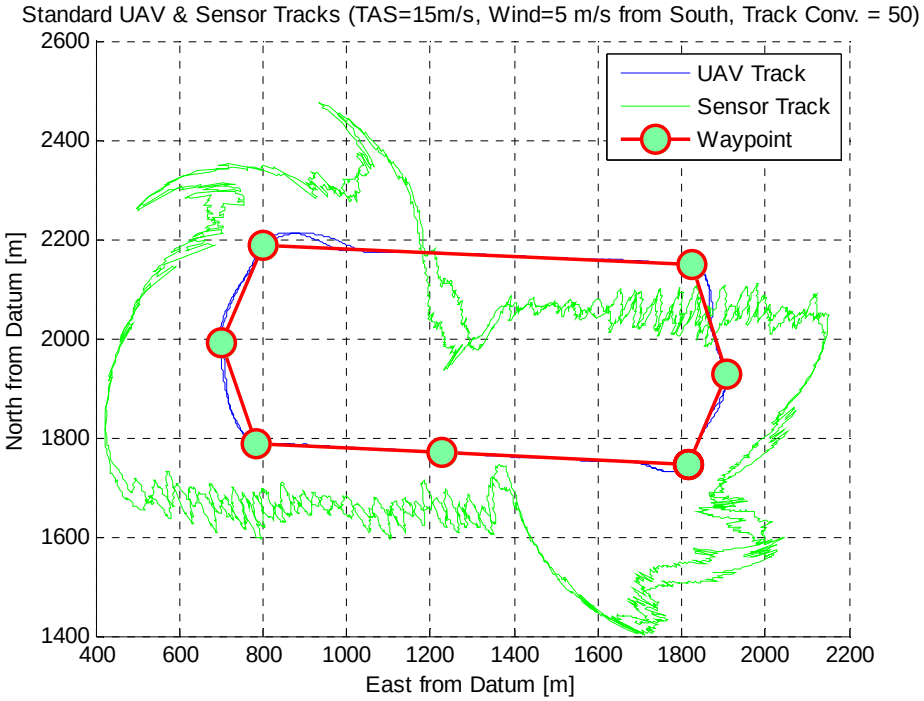

Figure 81. Standard UAV Race Track Pattern at 15 m/s with Wind=5 m/s and TC=50

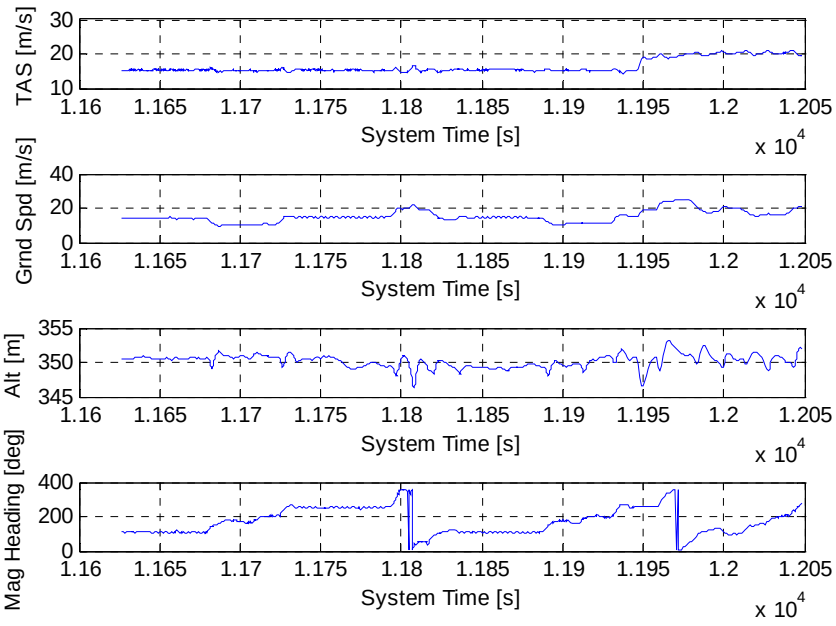

Figure 82. Various Parameters for the Race Track Pattern at 15 m/s, Wind5 m/s, & TC=50

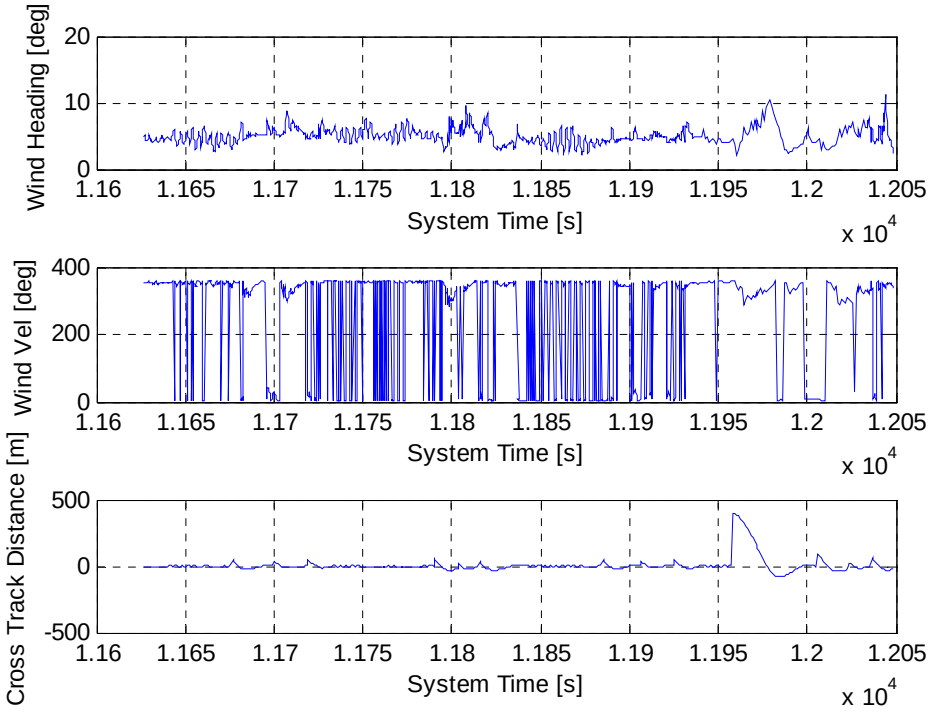

Figure 83. Real Time Wind Estimations for the Race Track at 15 m/s, Wind=5 m/s, & TC=50

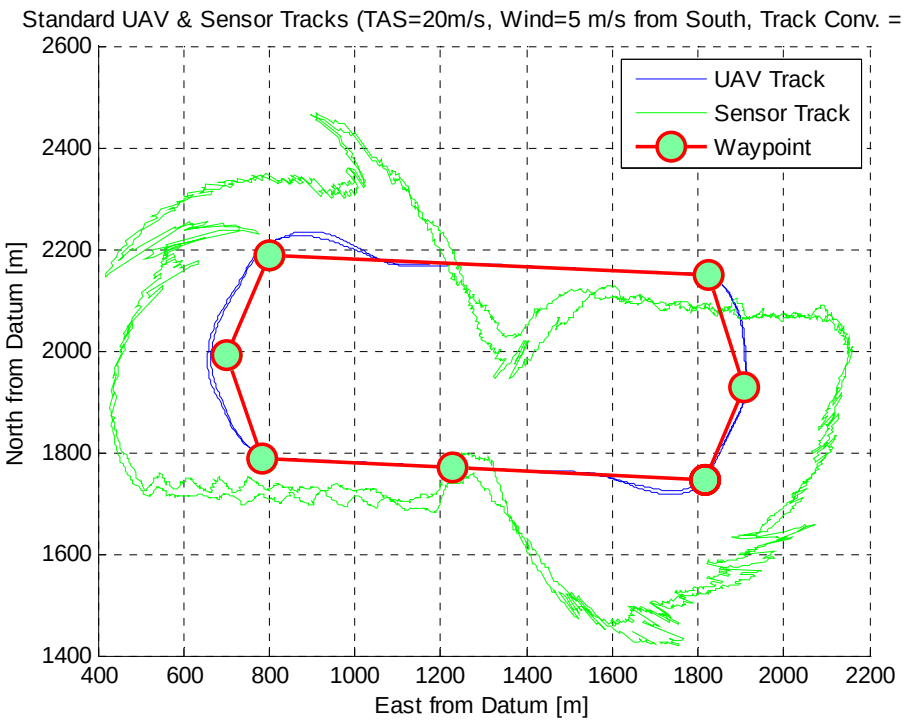

Figure 84. Standard UAV Race Track Pattern at 20 m/s with Wind=5 m/s and TC=50

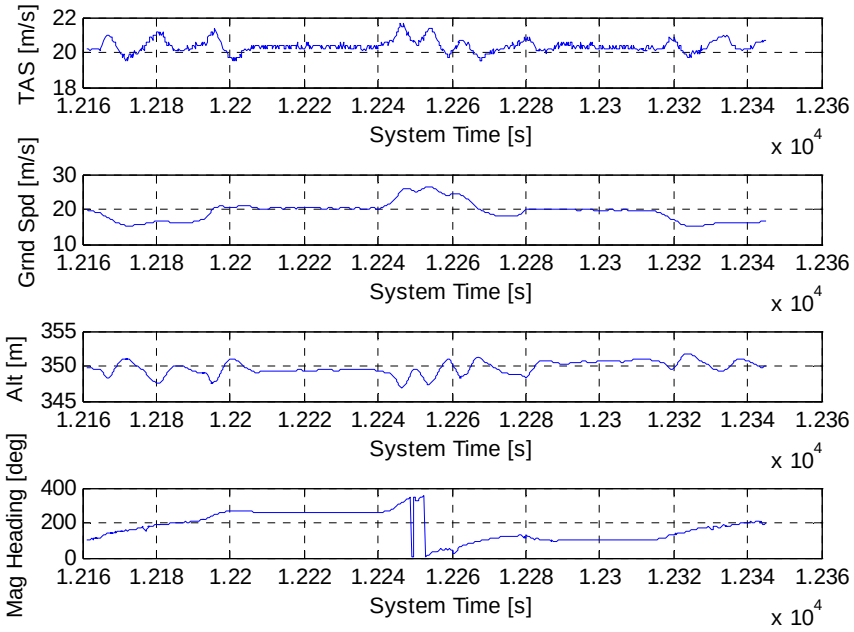

Figure 85. Various Parameters for the Race Track Pattern at 20 m/s, Wind5 m/s, & TC=50

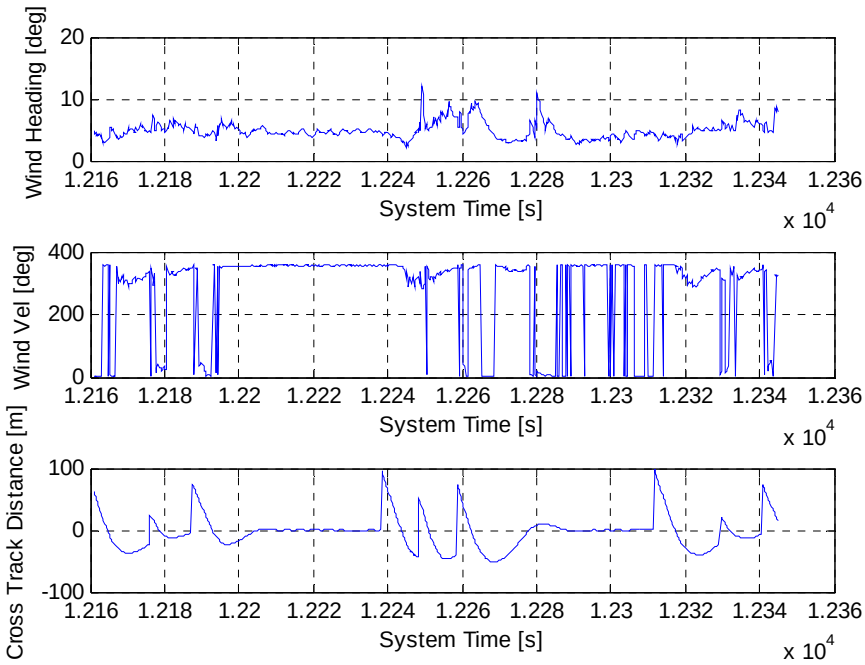

Figure 86. Real Time Wind Estimations for the Race Track at 20 m/s, Wind=5 m/s, & TC=50

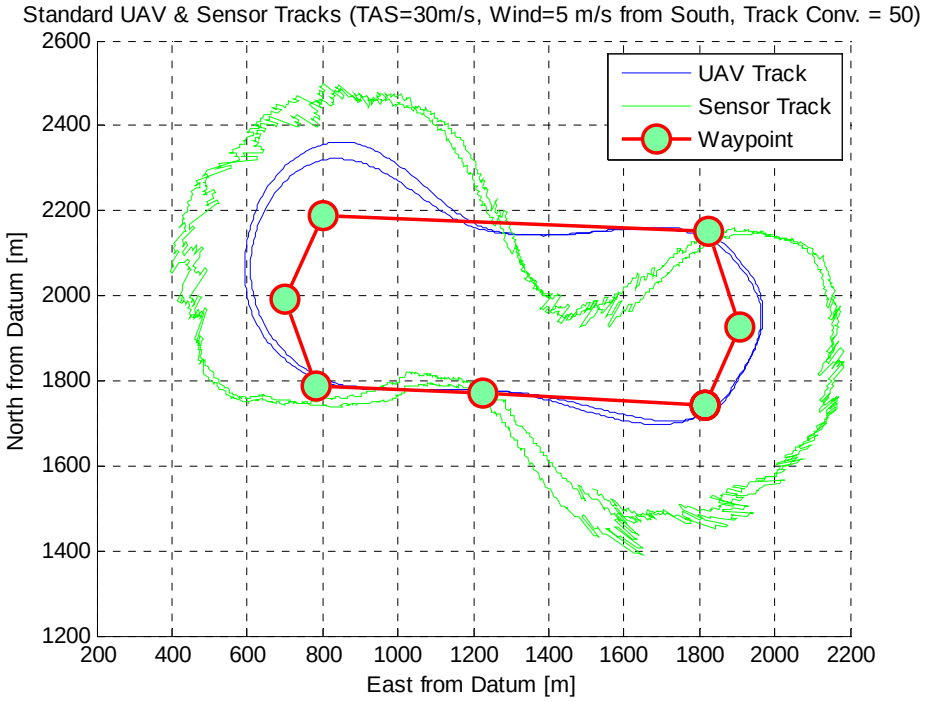

Figure 87. Standard UAV Race Track Pattern at 30 m/s with Wind=5 m/s and TC=50

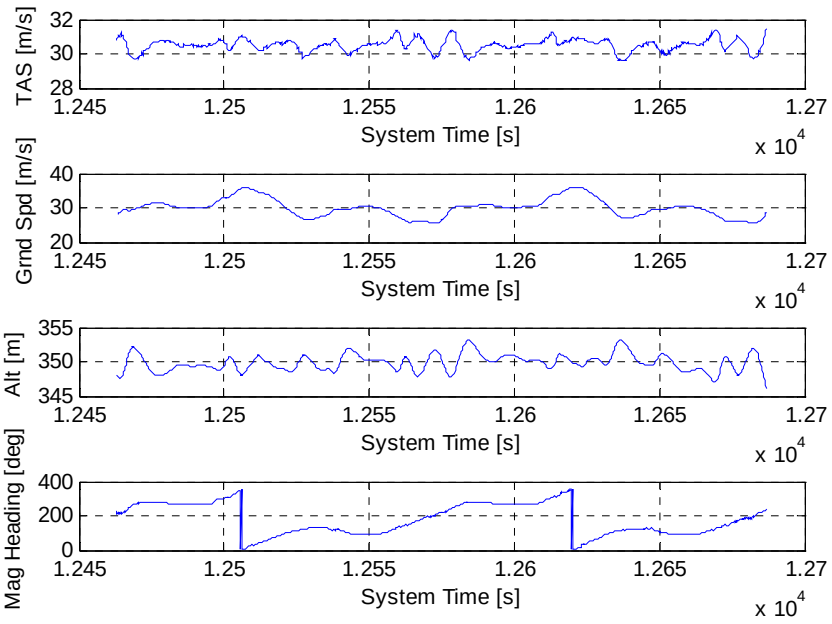

Figure 88. Various Parameters for the Race Track Pattern at 30 m/s, Wind5 m/s, & TC=50

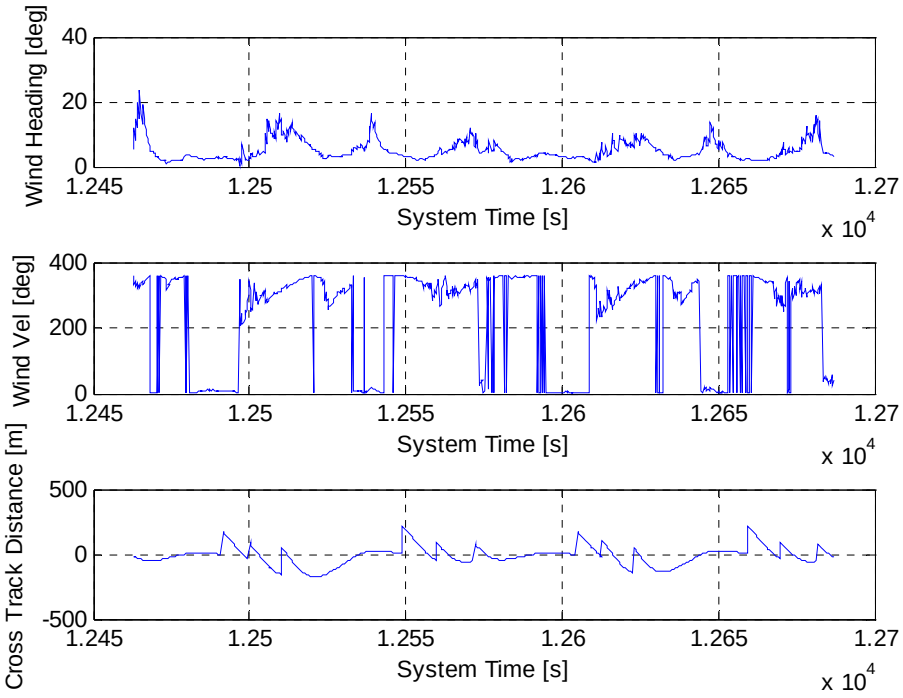

Figure 89. Real Time Wind Estimations for the Race Track at 30 m/s, Wind=5 m/s, & TC=50

MODIFIED FLIGHT PATH RESULTS

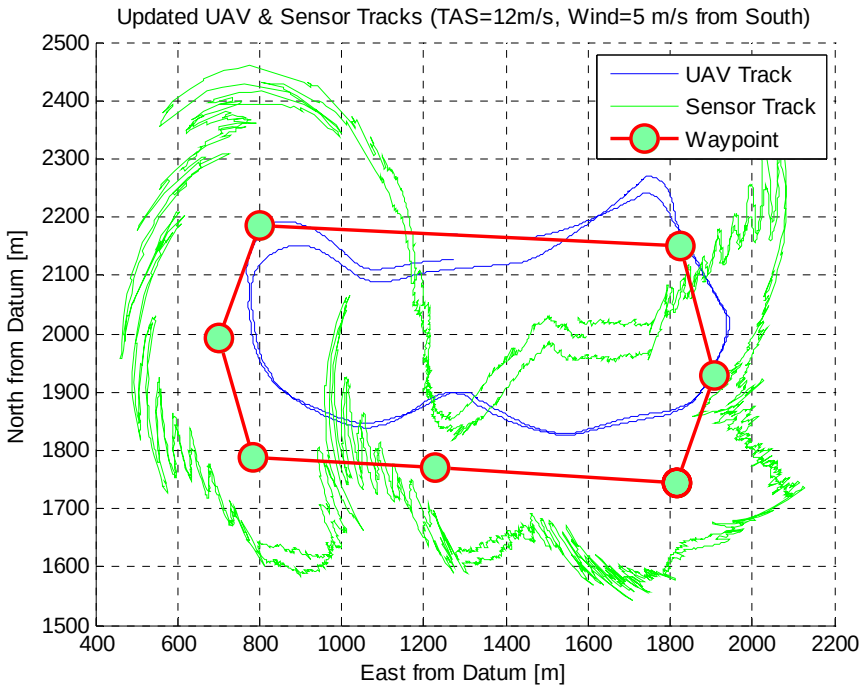

Figure 90. Updated UAV Race Track Pattern at 12 m/s with Wind=5 m/s and TC=250

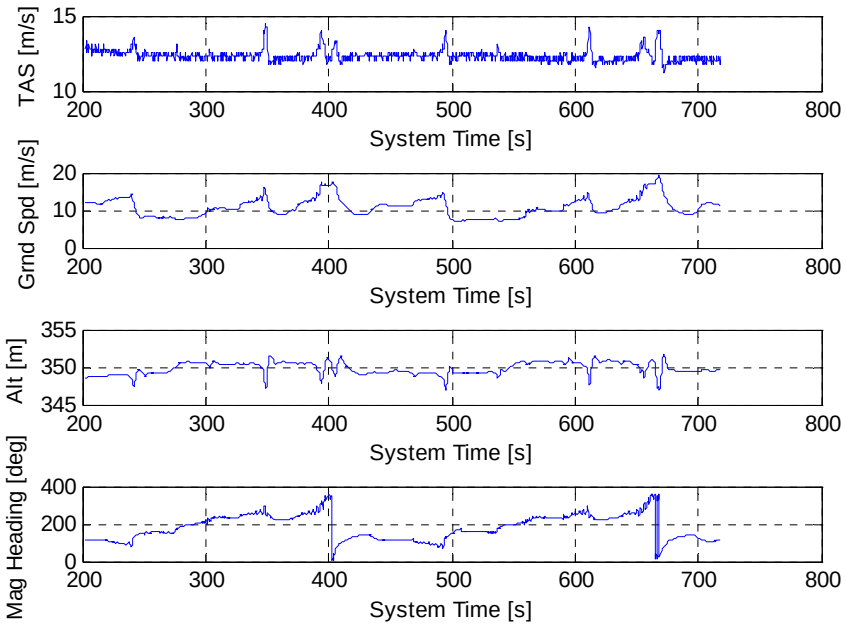

Figure 91. Various Parameters for the Race Track Pattern at 12 m/s, Wind5 m/s, & TC=250

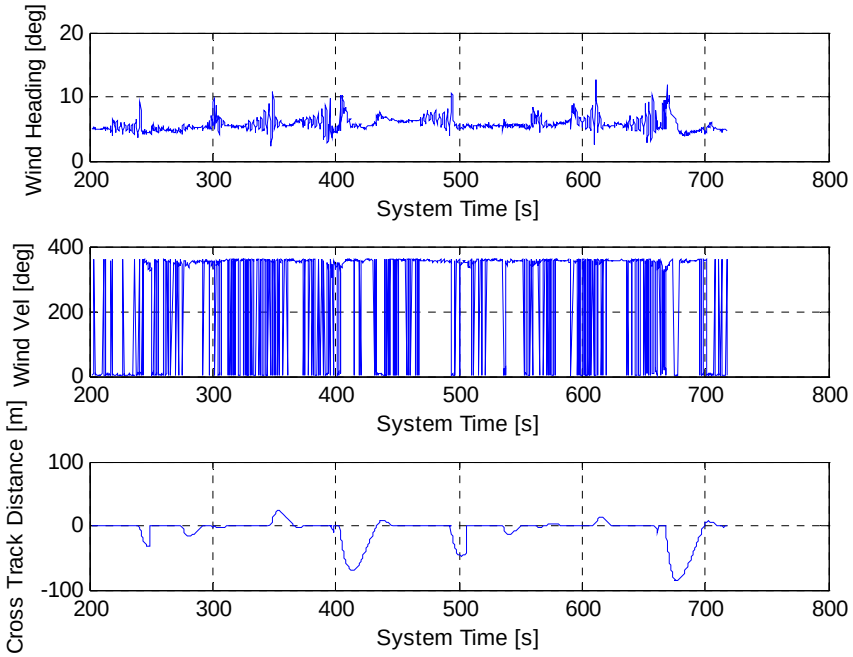

Figure 92. Real Time Wind Estimations for the Race Track at 12 m/s, Wind=5 m/s, & TC=250

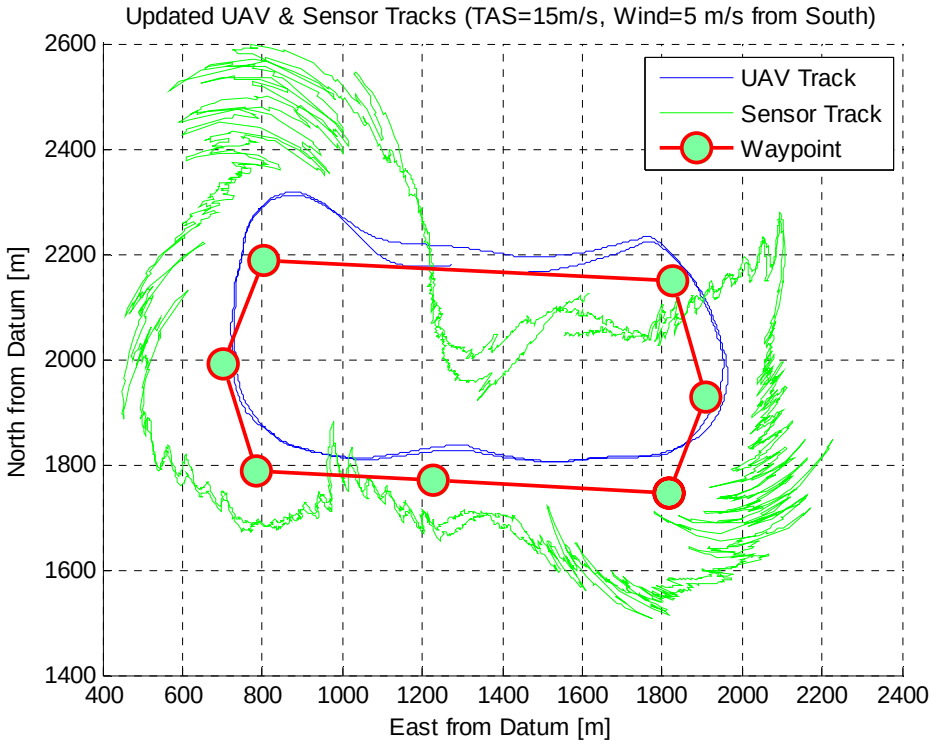

Figure 93. Updated UAV Race Track Pattern at 15 m/s with Wind=5 m/s and TC=250

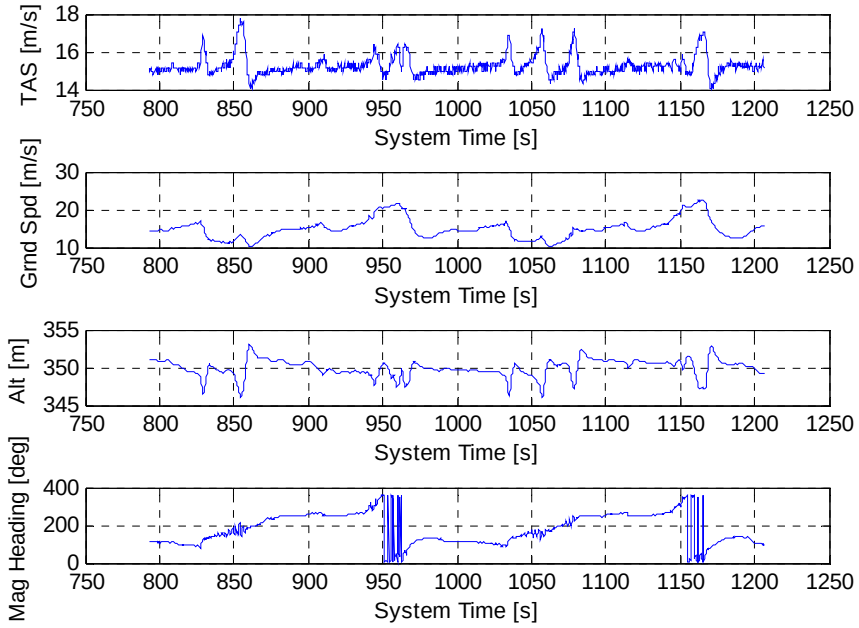

Figure 94. Various Parameters for the Race Track Pattern at 15 m/s, Wind5 m/s, & TC=250

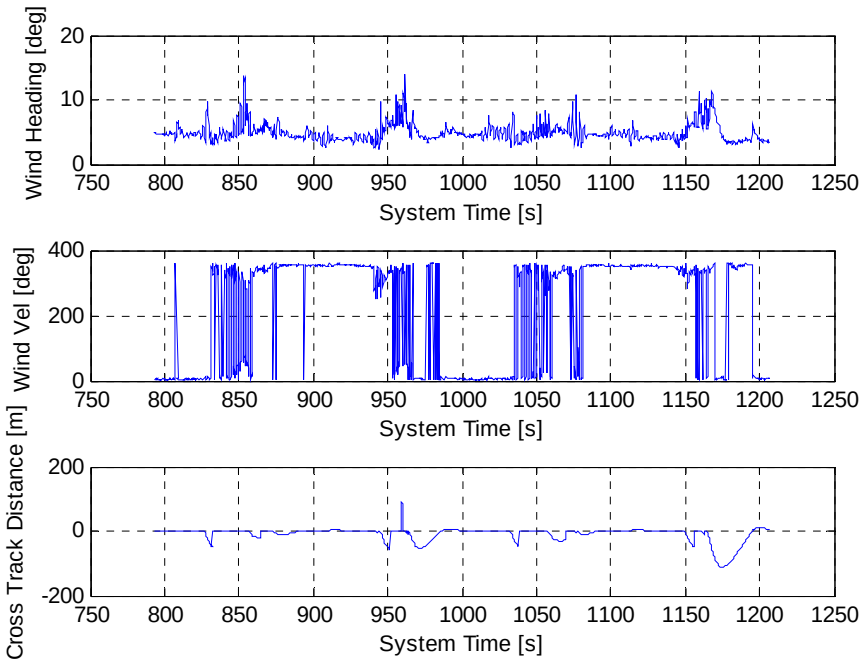

Figure 95. Real Time Wind Estimations for the Race Track at 15 m/s, Wind=5 m/s, & TC=250

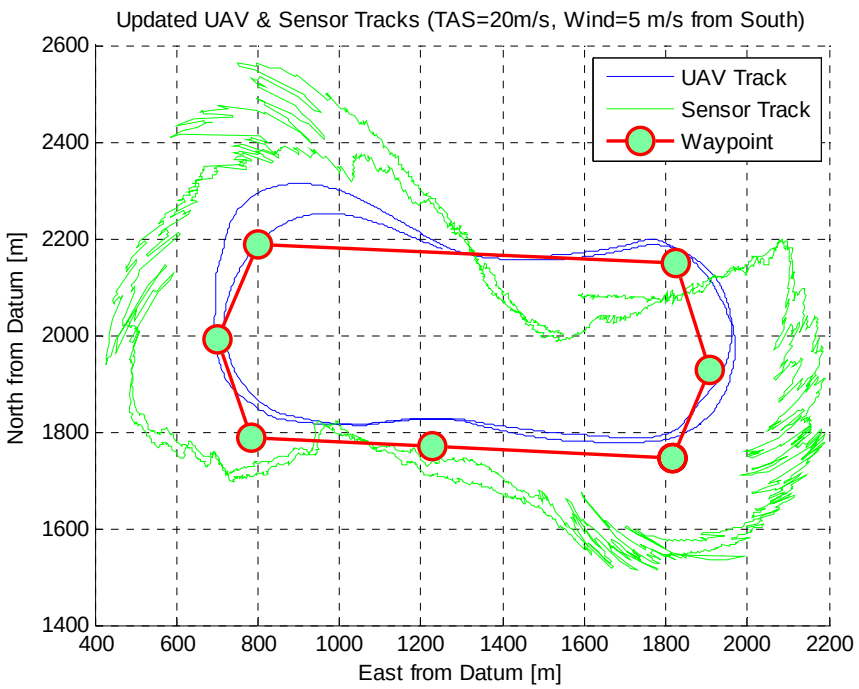

Figure 96. Updated UAV Race Track Pattern at 20 m/s with Wind=5 m/s and TC=250

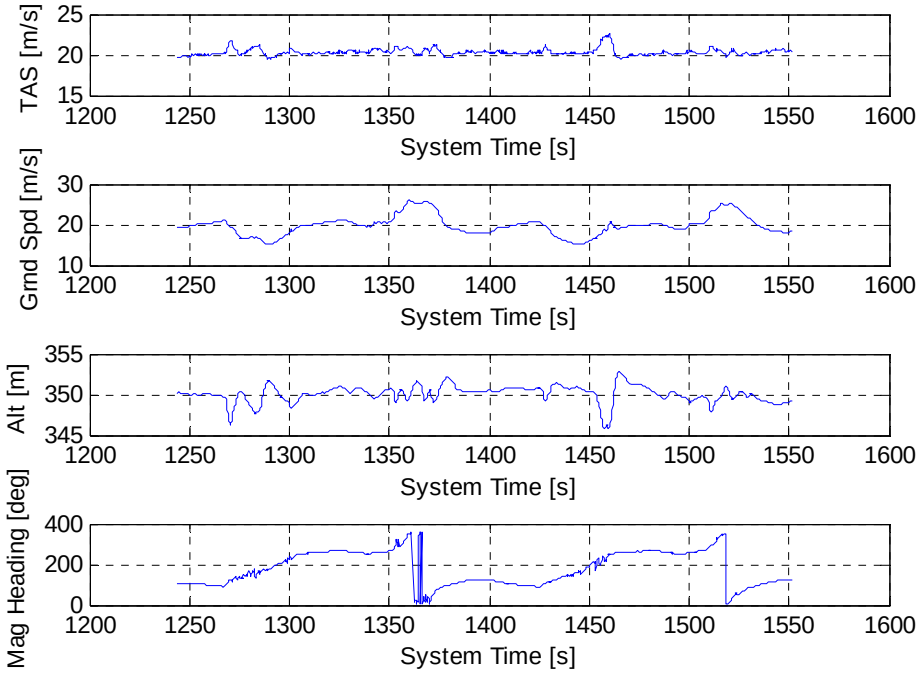

Figure 97. Various Parameters for the Race Track Pattern at 20 m/s, Wind5 m/s, & TC=250

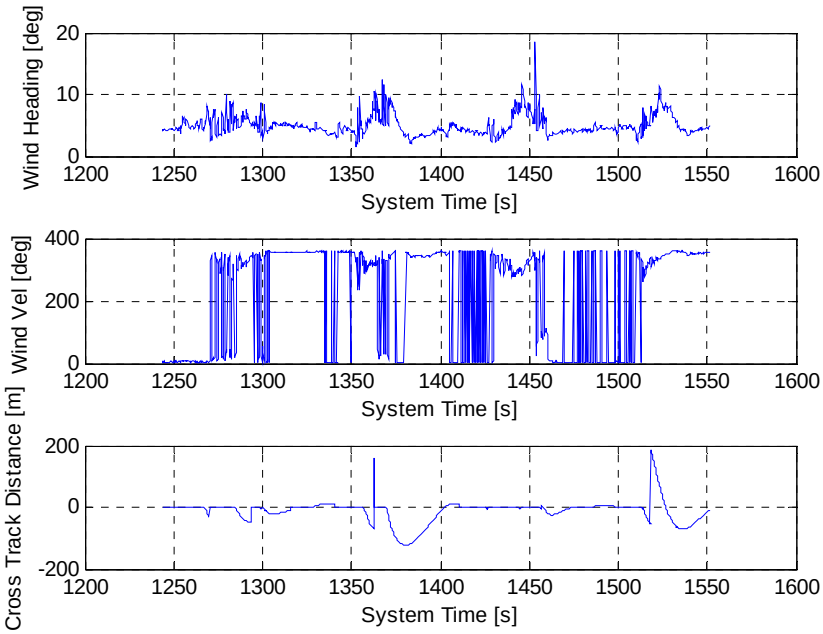

Figure 98. Real Time Wind Estimations for the Race Track at 20 m/s, Wind=5 m/s, & TC=250

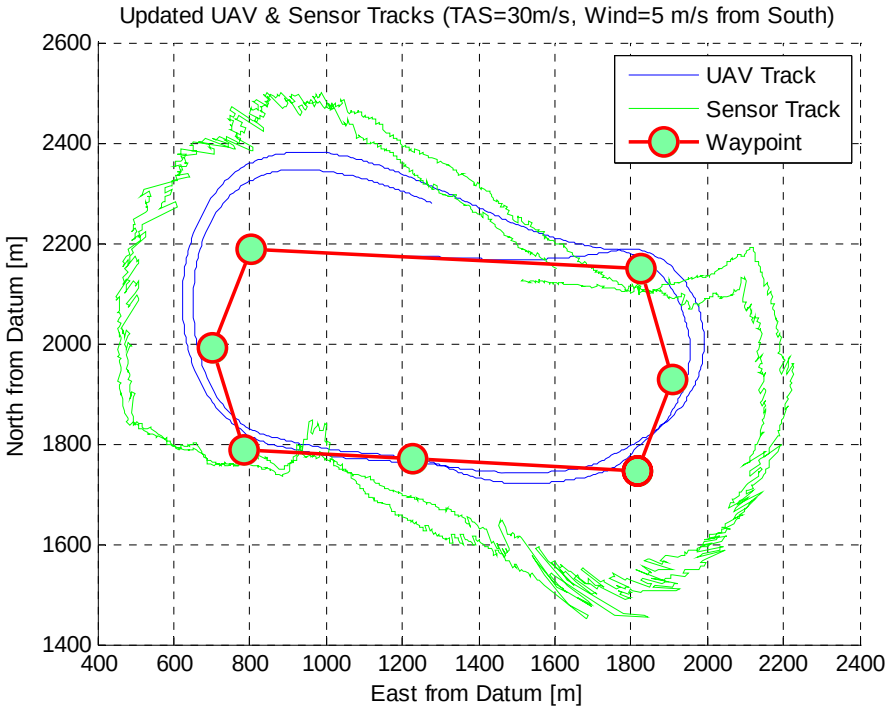

Figure 99. Updated UAV Race Track Pattern at 30 m/s with Wind=5 m/s and TC=250

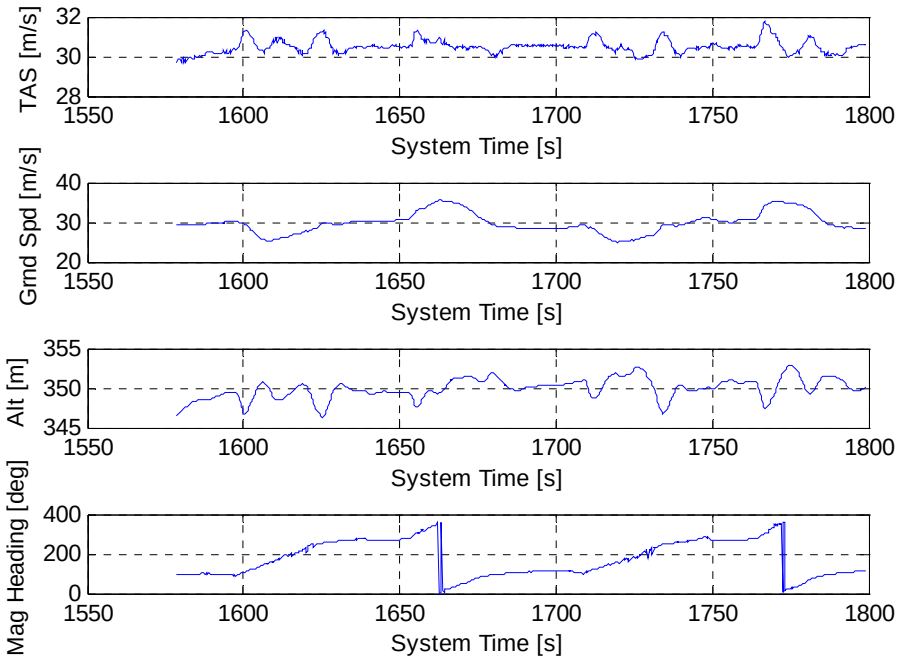

Figure 100. Various Parameters for the Race Track Pattern at 30 m/s, Wind5 m/s, & TC=250

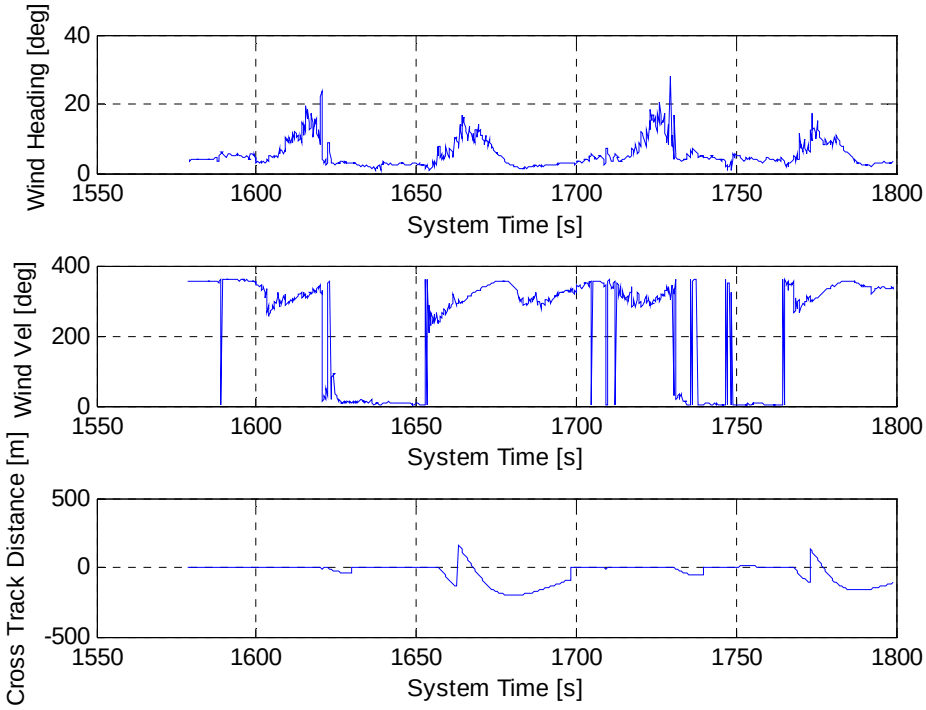

Figure 101. Real Time Wind Estimations for the Race Track at 30 m/s, Wind=5 m/s, & TC=250

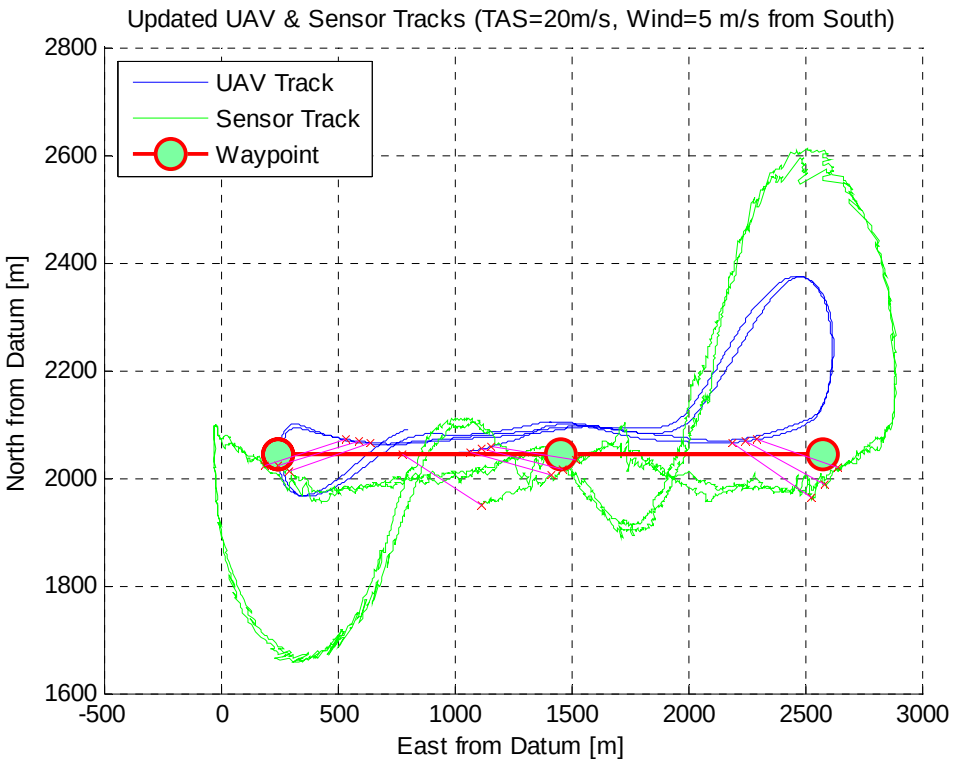

Figure 102. Updated Long Point to Point at 20 m/s with Wind=5 m/s and TC=250

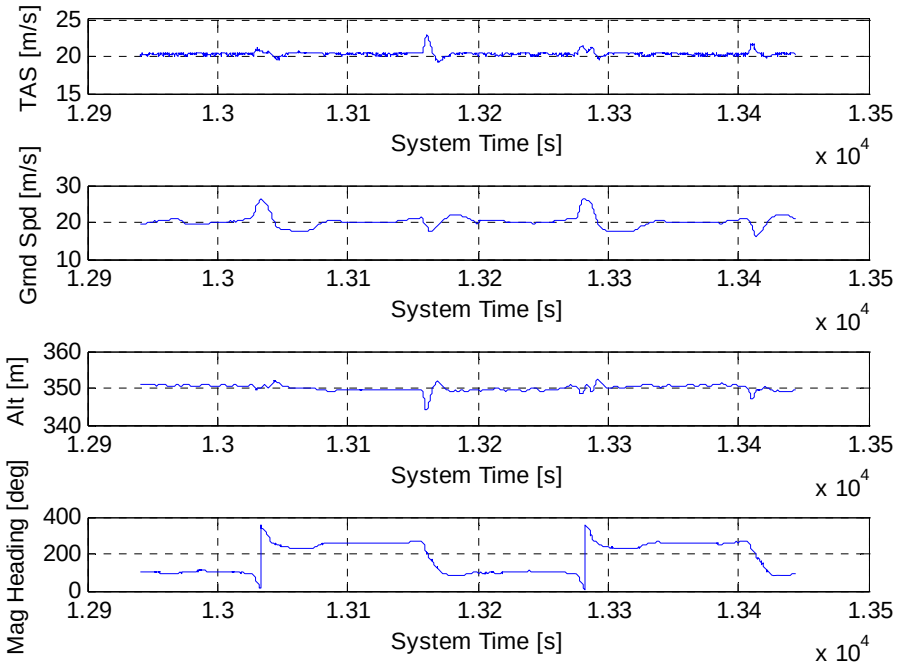

Figure 103. Various Parameters for the Long Point to Point at 20 m/s, Wind5 m/s, & TC=250

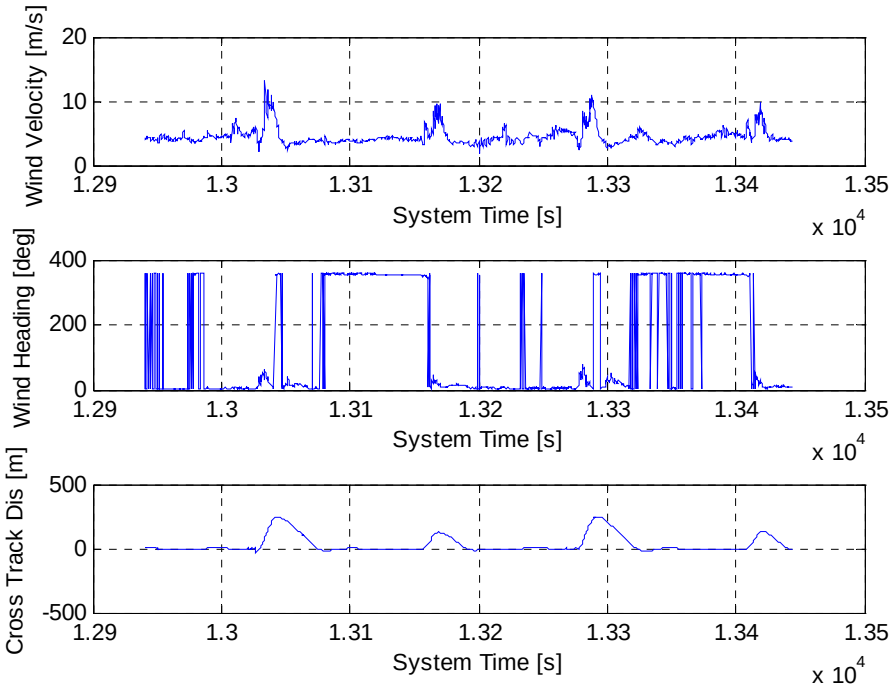

Figure 104. Real Time Wind Estimations for the Point to Point at 20 m/s, Wind=5 m/s, & TC=250

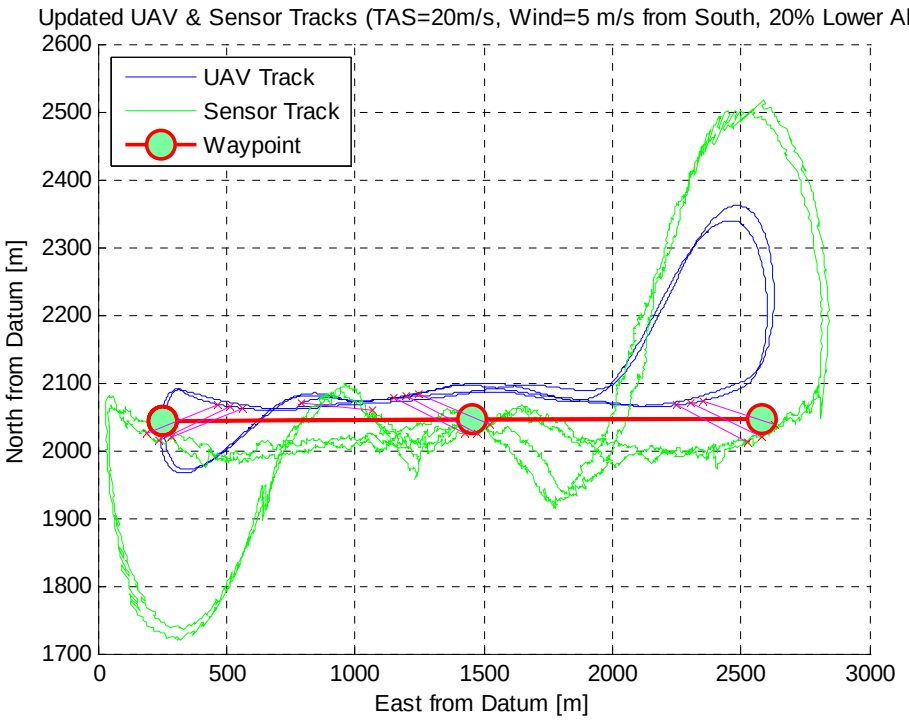

Figure 105. Updated Long Point to Point at 20 m/s with Wind=5 m/s & Lower Alt

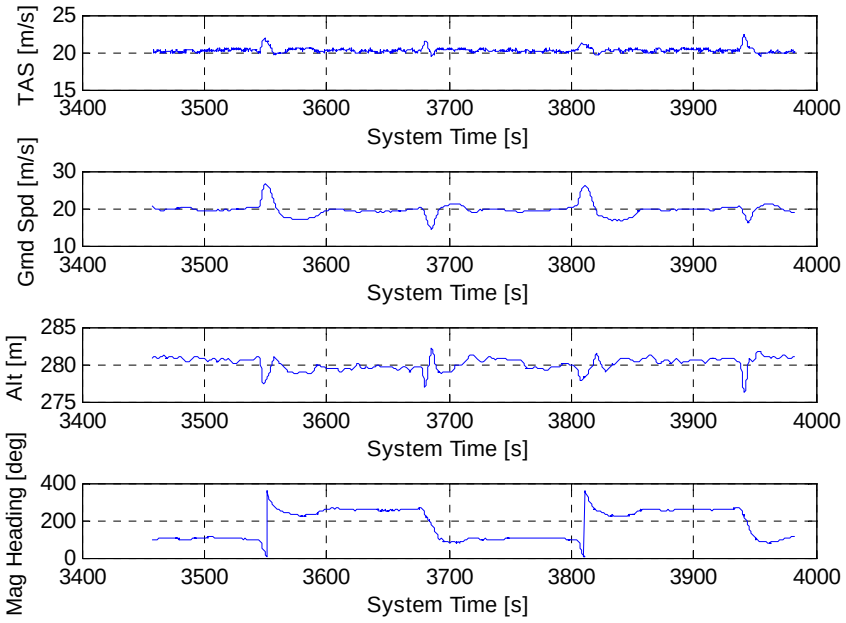

Figure 106. Various Parameters for the Long Point to Point at 20 m/s, Wind5 m/s, & Lower Alt

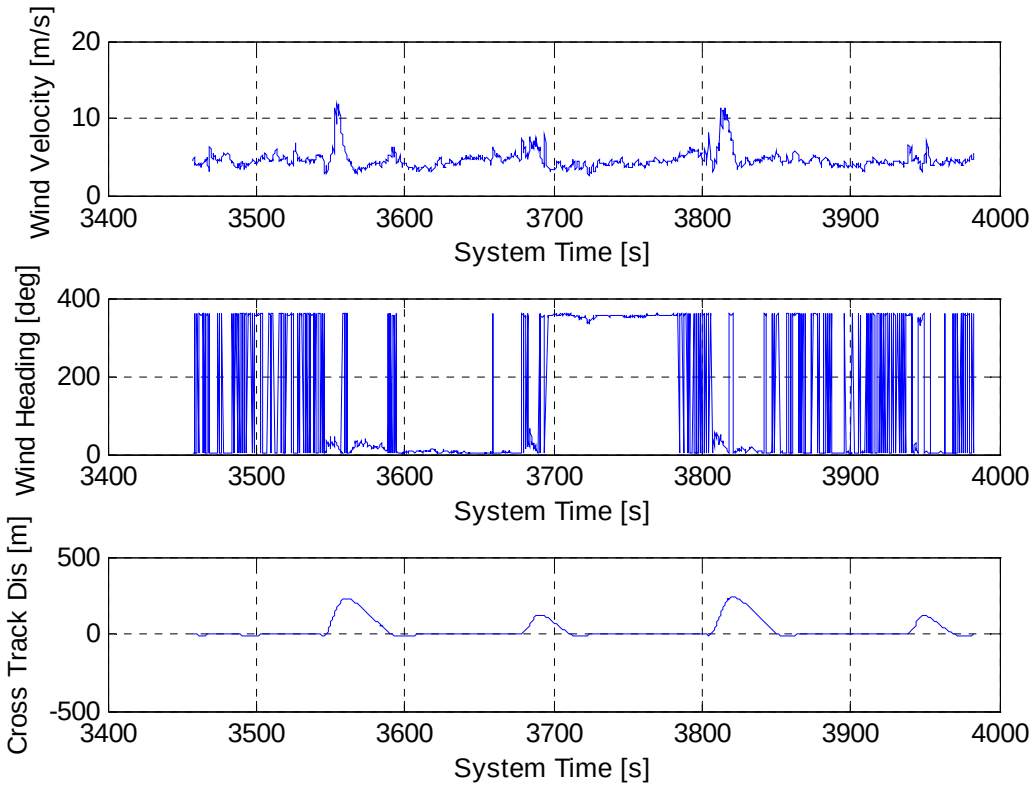

Figure 107. Real Time Wind Estimations for the Point to Point at 20 m/s, Wind=5 m/s, & Lower Alt

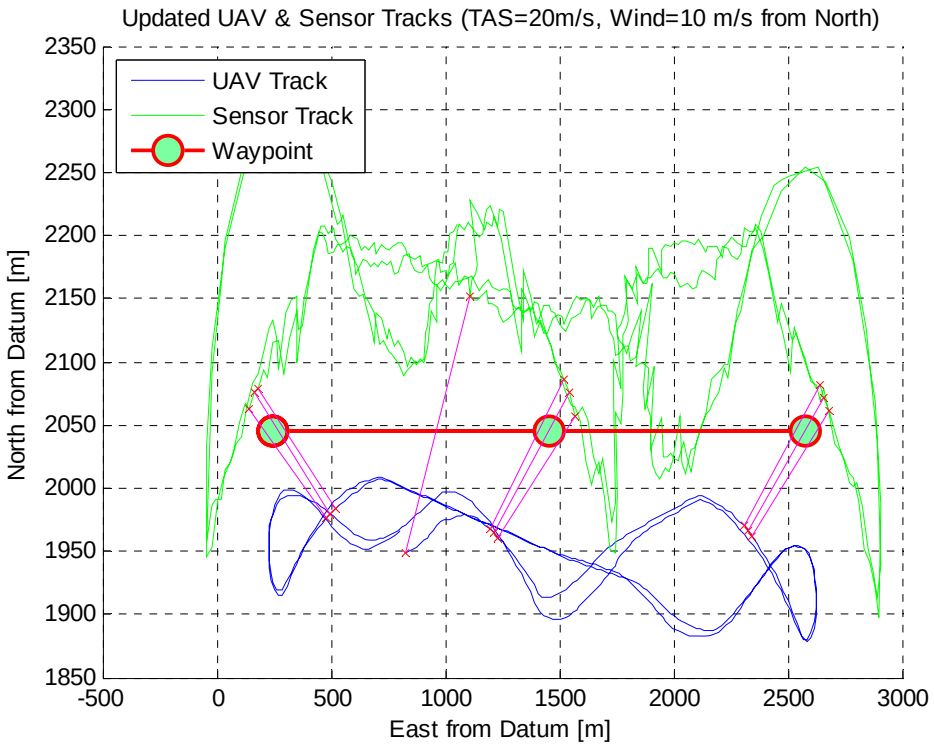

Figure 108. Updated UAV for Point to Point with Wind =10 from North

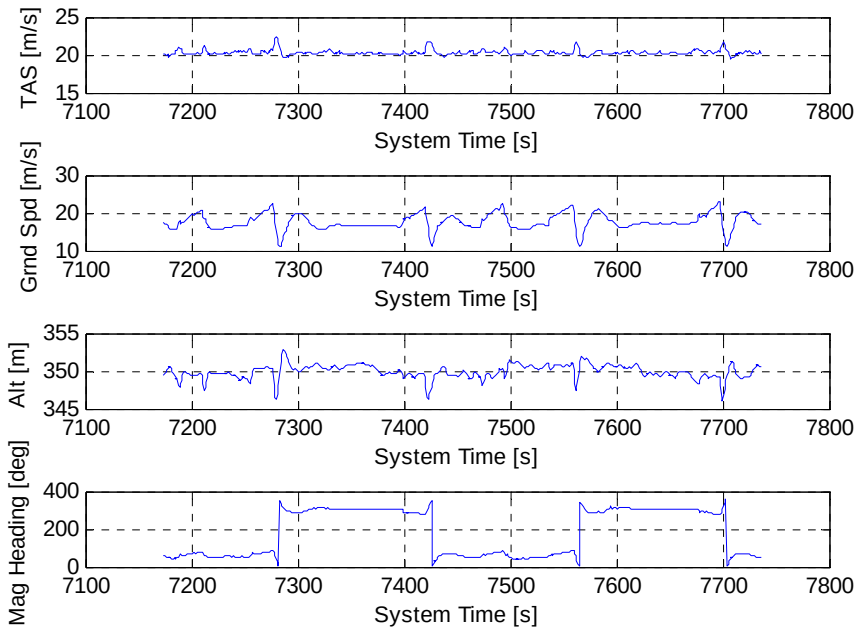

Figure 109. Various Parameters for the Point to Point with Wind=10 m/s from the North

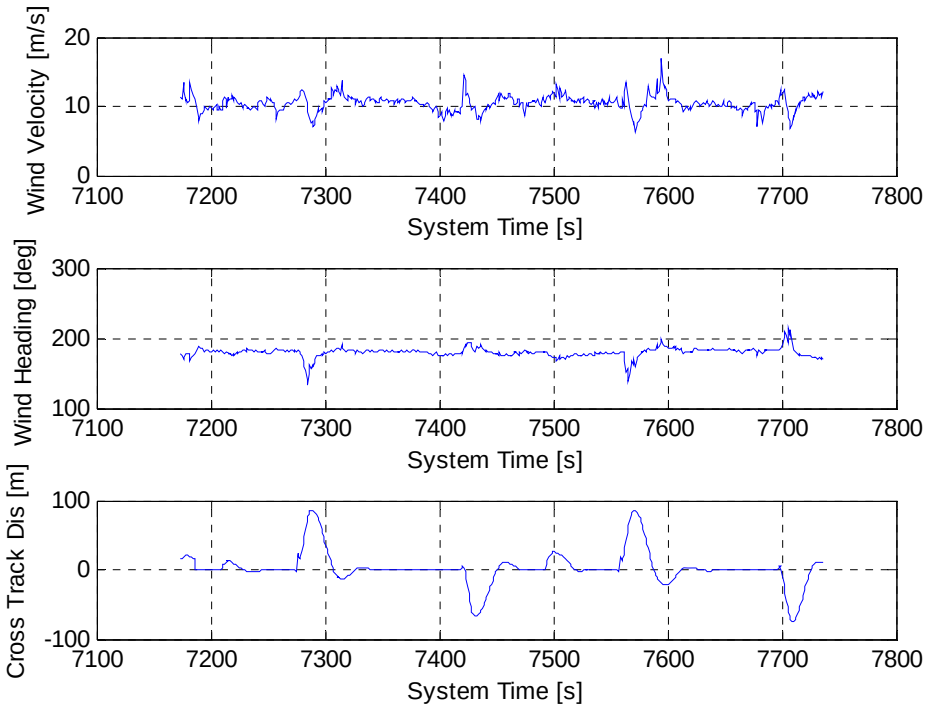

Figure 110. Real Time Wind Estimations for the Point to Point with the Wind=10 m/s from the North

Appendix B: Software Development Kit (SDK) C++ Code

```
/*****************************************************************************
Test file for piccolo communication

Programmed by: Brent Robinson

Date: 9 May, 2006

*****************************************************************************/

#include<iostream.h>
#include<conio.h>
#include<string>
#include "CommManager.h"
#include "Win32Serial.h"
#include<stdlib.h>
#include<windows.h>
#include"lla2enu.h"


using namespace std;

//Basepoint to use for all ENU coordinates...It's location is south and west of WPAFB
const double Base_X = 503000;
const double Base_Y = -4884700;
const double Base_Z = 4057800;


CCommManager* m_pComm = NULL;
Queue_t* pQ = NULL;

//Used for converting the waypoint lla data to ENU coords
ENUCoord PosENU;
ENUCoord newwpENU;
ENUCoord WayENU;

//To log the desired data in a .txt file
FILE * pFile1;


//function prototypes
void displayData(int i);

void BrentsWindCorrection(int i);                       //Real time wind estimation function
void SensorAdjustment(int i);                           //Wind Corrected Sensor Pointing
//void HeadingAdjust(int i);                            //Heading Adjustment function for the "turn rate approach"
//void AirspeedAdjust(int i);                           //Airspeed Adjustment function for the "turn rate approach"
//void WaypointTrackReturn(int i);                      //Attempt at a function to turn off the turn rate commanding and return to normal
ops
//void WaypointFlyingnotTrackFlying(int i);             //Attempt at a function making the Piccolo a pure waypoint tracker instead
of a track follower
//void WaypointInfoFinding(int i, FPPoint_t& pntWaypoint);      //Attempt at a function which accesses the waypoint lla data
//void UpdatingWaypointadjustingforWind(int i);//, int next);   //Updating "Rabbit" approach




//data structure to hold telemetry packet data
typedef struct
{
        double Longitude;           //from LLA data: Telemtry packet
        double Latitude;            //from LLA data: Telemtry packet
        double East;                        //calculated from LLA data using lla2enu class
        double North;                       //calculated from LLA data using lla2enu class
        double Up;                  //calculated from LLA data using lla2enu class
        float Altitude;                     //from LLA data: Telemtry packet
        float Velocity;                     //from GPS.Speed: Telemetry packet
//      float Alpha                                 //anlge between velocity and direction of nose of plane vertically
```

```
//              float Beta;                             //angle between velocity and direction of nose of plane horizantally
                int Hours;
                int Minutes;
                float Seconds;

   //Brent added these
                float Direction;            //GPS Ground Track Direction
                float TAS;                  //Aircraft TAS
   double Lat;                  //Current aircraft Latitude
                double Lon;                 //Current aircraft Longitude
                float CrossTrack;           //Current Cross Track Distance
                float AlongTrack;           //Current Along Track Distance - Distance from the current waypoint
} telemetry;


//data structure to hold control packet data
typedef struct
{
                float Heading;                          //from Yaw reading: Control Data packet
                float BankAngle;                //from Roll: Control Data packet
                float RollRate;                         //from Roll Rate: Control Data packet
                float PitchRate;                //from Pitch Rate: Control Data packet
                float YawRate;                          //from Yaw Rate: Control Data packet

                float Aileron;
                float Elevator;
                float Throttle;
                float Rudder;
                int Hours;
                int Minutes;
                float Seconds;

                //Brent added these
                float MagHeading;           //Current aircraft magnetic heading
                float Pdynamic;             //Current dynamic pressure
} control;


// global variable used when the packets are decoded - allows for 10 networks
telemetry current_telemetry[10];
control current_control[10];

//Brent added these
FPPoint_t current_waypoint[10];   //Attempt at setting up another "switch" group
Gains_t current_gains[10];        //Attempt at setting up another "switch" group


//Brent ADDED these
float V_w;
float Chi_w;
float Chi_w_deg;
float V_TASnew;
float density;
float Pdynamic_new;
float Chi_Magnew;
float Chi_Magnew_deg;
float turnrate;
float cmd_speed;
int count=0;
float e;
float f;
float Dis;
//float toofar;
//float angle_deg;
//float angle;
//float abscos;
//float abssin;
float enu69east;
float enu69north;
double current_waypoint_Latitude;
```

```
double current_waypoint_Longitude;
float current_waypoint_Altitude;
UInt8 Waypoint_cmd[10];
UInt8 orig;
UInt8 orignext;

float Dis2;
float Horiz;
double Adjust1;
double point0Lat;
double point0Lon;
double Alt0;
double point1Lat;
double point1Lon;
double Alt1;
double point2Lat;
double point2Lon;
double Alt2;
double point3Lat;
double point3Lon;
double Alt3;
double point4Lat;
double point4Lon;
double Alt4;
double point5Lat;
double point5Lon;
double Alt5;
double point6Lat;
double point6Lon;
double Alt6;

double enu60east;
double enu60north;
double enu60alt;
double enu61east;
double enu61north;
double enu61alt;
double enu62east;
double enu62north;
double enu62alt;
double enu63east;
double enu63north;
double enu63alt;
double enu64east;
double enu64north;
double enu64alt;
double enu65east;
double enu65north;
double enu65alt;
double enu66east;
double enu66north;
double enu66alt;

float EastonTrack;
float NorthonTrack;
float e2onTrack;
float f2onTrack;
float LOS_Dis;

float Adjust1a;
float Adjust2a;
float T;
float theta_one;
float e2;
float f2;

float star;
float sinfromNext;
float cosfromNext;
```

```
float theta_m;
float delta_1;
float delta_2;



//clears the screen
void clrscr()
{
  HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
  COORD coord = {0, 0};
  DWORD count;

  CONSOLE_SCREEN_BUFFER_INFO csbi;
  GetConsoleScreenBufferInfo(hStdOut, &csbi);

  FillConsoleOutputCharacter(hStdOut, ' ', csbi.dwSize.X * csbi.dwSize.Y, coord, &count);

  SetConsoleCursorPosition(hStdOut, coord);
}

//as defined in "index.html": from SDK documentation
void NewNetwork(UInt16 NetworkID, void* Parameter)
{

}

//looks for and gleans data from an autopilot packet sent from a network
void LookForAutopilotData(QType* pQ, int whosData)
{
   static AutopilotPkt_t APPkts[10];

            UInt32 i, NumNets;
            SInt32 ID;

            //look at how many networks m_pComm can see
            NumNets = m_pComm->GetNumNets();

            for(i = 0; i < NumNets; i++)
            {
                        // Don't display past 10 networks since we didn't include the space
                        if(i >= 10) break;

                        ID = m_pComm->GetIDFromIndex(i);

                        // Don't try to decode ground station packets
                        //if(ID < 1) continue;

                        // Get the pointer to the receive queue for the autopilot stream.  Note
                        //   this pointer will persist as long as the network structure exists,
                        //   so we could just save the pointer and then we wouldn't have the
                        //   overhead of repeatedly calling this function
                        pQ = m_pComm->GetStreamRxBuffer((UInt16)ID, AUTOPILOT_STREAM);

                        if(!pQ) continue;

                        // Now check to see if a packet exists.  Note!!! The raw packet
                        //   structure MUST persist between calls, and it MUST be unique to this
                        //   network.


                        if(LookForAutopilotPacket(pQ, &(APPkts[i])))
                        {
                                    switch(APPkts[i].PktType)
                                    {
                                                case TELEMETRY:
                                                            UserData_t telemData;
                                                            DecodeTelemetryPacket(&(APPkts[i]), &(telemData));
                                                            //update telemtry struct
```

```
                                        current_telemetry[i].Longitude = telemData.GPS.Longitude * 180.0 /
3.1415926;
                                        current_telemetry[i].Latitude = telemData.GPS.Latitude * 180.0 / 3.1415926;
                                        current_telemetry[i].Altitude = telemData.GPS.Altitude;
                                        current_telemetry[i].Velocity = telemData.GPS.Speed;
                                    current_telemetry[i].Direction = telemData.GPS.Direction;      //Brent added
            current_telemetry[i].TAS = telemData.TAS;                   //Brent added
                                        current_telemetry[i].CrossTrack = telemData.CrossTrack;        //Brent added
                                        current_telemetry[i].AlongTrack = telemData.AlongTrack;        //Brent added



                                        //convert lla data to enu
                                        PosENU.lla2enu(current_telemetry[i].Latitude *3.1415926/180,
                                                                    current_telemetry[i].Longitude
*3.1415926/180,
                                                                    current_telemetry[i].Altitude,
                                                                    Base_X, Base_Y, Base_Z);

                                        current_telemetry[i].East = PosENU.GetEast();
                                        current_telemetry[i].North = PosENU.GetNorth();
                                        current_telemetry[i].Up = PosENU.GetUp();

                                        current_telemetry[i].Hours = telemData.GPS.hours;
                                        current_telemetry[i].Minutes = telemData.GPS.minutes;
                                        current_telemetry[i].Seconds = telemData.GPS.seconds;
                                        //display the data

                                        //Brent added...This is all the data that is written to a log file
                                        fprintf(pFile1,"\n %i    %i     %i      %f    %f    %f    %f    %f     %f
%f          %f          %f     %f       45         %f        %f", ID,
                                                                    current_control[i].Hours, current_control[i].Minutes,
                                                                    current_control[i].Seconds,
current_telemetry[i].CrossTrack,

            current_telemetry[i].Velocity,(current_telemetry[i].Direction*(180/3.1415926)),
                                                                    current_telemetry[i].TAS,
current_control[i].MagHeading,
                                                                    V_w,
Chi_w_deg,(current_telemetry[i].Direction*(180/3.1415926)-current_control[i].MagHeading),

            current_telemetry[i].Altitude,(current_telemetry[i].Altitude/cos((45*(3.1415926/180)))),sqrt(((current_telemetry[i].Altitud
e/cos((45*(3.1415926/180))))*(current_telemetry[i].Altitude/cos((45*(3.1415926/180)))))-
(current_telemetry[i].Altitude*current_telemetry[i].Altitude)));

                                        displayData(whosData);
                                        break;


    //Brent's attempt at accessing the waypoint lla data
                                    case WAYPOINT:
                                {
                                    //Waypoint_t wayData;
                                        //UserData_t wayData;                   //Brent
                                        FPPoint_t wayData;                      //THIS IS THE FIRST PLACE
                                    //      wayData.Point.Lat = 0.0;
                                    //      wayData.Point.Lon = 0.0;
                                    //      wayData.Point.Alt = 0.0;
                                        UInt8 This = 0;
                                        //This = DecodeWaypointPacket(&(APPkts[i]), &(wayData));    //WHERE I
TRY TO GET THE WAYPOINT LAT/LONG
                                        This = DecodeWaypointPacket(&(APPkts[i]), &(wayData));    //WHERE I
TRY TO GET THE WAYPOINT LAT/LONG

                                        /*          if (Waypoint_cmd[i] != 69 || Waypoint_cmd[i] != 70)
                                                {
                                                        WaypointInfoFinding(i, wayData);
                                                }
                                        */
```

```
                                        //              FPPoint_t Point;
                                        //              UInt8 This = 0;
                                        //              This = DecodeWaypointPacket(&(m_APPkts[i]), &Point);

                                }
                                                displayData(whosData);
                                                break;

                        //Brent added...This allows a variable "Waypoint_cmd" that is the index of the current waypoint being tracked
                                                case AUTOPILOT_COMMAND:
        AutopilotCmd_t Cmd[3];
                                                Waypoint_cmd[i] = DecodeAutopilotControlPacket(&(APPkts[i]), &Cmd[i]);
                                                displayData(whosData);
                                                break;


                                                case CONTROL_DATA:
                                                        UserData_t controlData;
                                                        float gyroBias[3], controls[10];
                                                        DecodeControlDataPacket(&(APPkts[i]), &(controlData), gyroBias, controls);
                                                        //update telemetry struct
                                                        current_control[i].BankAngle = controlData.Euler[0] * 180/3.1415926;
                                                        current_control[i].Heading = controlData.Euler[2] * 180/3.1415926;
                                                                //Euler[0] = Rroll, Euler[1] = Pitch, Euler[2] = Yaw
                                                        current_control[i].RollRate = controlData.Gyro[0] * 180/3.1415926;
                                                        current_control[i].PitchRate = controlData.Gyro[1] * 180/3.1415926;
                                                        current_control[i].YawRate = controlData.Gyro[2] * 180/3.1415926;

                                                        current_control[i].Aileron = controls[0] * 180/3.1415926;
                                                        current_control[i].Elevator = controls[1] * 180/3.1415926;
                                                        current_control[i].Throttle = controls[2];
                                                        current_control[i].Rudder = controls[3] * 180/3.1415926;
                                                        //convert GPS seconds into hours, minutes, and seconds
                                                        double hours = controlData.SystemTime / 3600000.0;
                                                        current_control[i].Hours = hours;
                                                        double mins = (hours - (double)current_control[i].Hours) * 60;
                                                        current_control[i].Minutes = mins;
                                                        current_control[i].Seconds = (mins - (double)current_control[i].Minutes) * 60;

                                                        //Brent added these
        current_control[i].Pdynamic = controlData.Pdynamic;
        current_control[i].MagHeading = controlData.MagHeading * 180/3.1415926;

                                                        displayData(whosData);   //display the data
                                                        break;
                                }
                        }
                }
}// LookForAutopilotData


//prints the telemetry, control, and obstacle avoidance data to screen
void displayData(int i)
{
        //print current telemetry data
        clrscr();
        printf("ID = %i", m_pComm->GetIDFromIndex(i));

        printf("\nTelemetry Packet Data  : %i", current_telemetry[i].Hours);
        printf(":%i", current_telemetry[i].Minutes);
        printf(":%f", current_telemetry[i].Seconds);
        printf("\nLatitude (deg)         : %f", current_telemetry[i].Latitude);
        printf("                  East: %f", current_telemetry[i].East);
        printf("\nLongitude (deg)        : %f", current_telemetry[i].Longitude);
        printf("                  North: %f", current_telemetry[i].North);
        printf("\nAltitude (m)           : %f", current_telemetry[i].Altitude);
        printf("                  Up: %f", current_telemetry[i].Up);
        printf("\nGround Speed           : %f", current_telemetry[i].Velocity);
        printf("\n\nControl Packet Data  : %i", current_control[i].Hours);
        printf(":%i", current_control[i].Minutes);
```

```
		printf(":%f", current_control[i].Seconds);
		printf("\nHeading		: %f", current_control[i].Heading);
//		printf("\nBank Angle		: %f", current_control[i].BankAngle);
//		printf("\nRoll Rate		: %f", current_control[i].RollRate);
//		printf("\nPitch Rate		: %f", current_control[i].PitchRate);
//		printf("\nYaw Rate		: %f", current_control[i].YawRate);



  //Brent added to be displayed
   printf("\nUAV TAS		: %f", current_telemetry[i].TAS);
		printf("\n\nUAV GPS DIRECTION	: %f", current_telemetry[i].Direction*180/3.14159);
		printf("\nUAV MAG HEADING	: %f", current_control[i].MagHeading);
//		printf("\nBRENTS BETA 2		: %f", current_telemetry[i].Direction*180/3.14159-current_control[i].MagHeading);
		printf("\nBrents WIND VELOCITY (m/s)  : %f", V_w);
		printf("\nBrents WIND DIRECTION  : %f", Chi_w_deg);
		printf("\nBrents NEW TAS		: %f", V_TASnew);
		printf("\nBrents NEW Mag Head	: %f", Chi_Magnew_deg);
		printf("		Brents pdyn	: %f", current_control[i].Pdynamic);
		printf("		Brents pdyn new  : %f", Pdynamic_new);
//  printf("\nBrents density		: %f", density);
		printf("\nWaypoint index		: %i", Waypoint_cmd[i]);
		printf("\nAdjust1		: %f", Adjust1a);
		printf("\nAdjust2		: %f", Adjust2a);
		printf("\nBrents Cross Track	: %f", current_telemetry[i].CrossTrack);
//		printf("\n along track		: %f", current_telemetry[i].AlongTrack);
		printf("\nDistance to Wypt	: %f", Dis2);
//		printf("\nTURNRATE :		%f",turnrate);
//		printf("\nWaypoint Lon		: %f", current_waypoint_Longitude);
//		printf("\nWaypoint Lat		: %f", current_waypoint_Latitude);
//		printf("\nWaypoint Alt		: %f", current_waypoint_Altitude);
//		printf("\nNew Waypoint Lat	: %d", newwpENU.GetLat());
//		printf("\nNew Waypoint Lon	: %d", newwpENU.GetLong());
//		printf("\ntheta_one		: %f",theta_one);

}//displayData



//This is the wind finding and subsequent new heading and airspeed function
void BrentsWindCorrection(int i)
{
//These are the basic vector equations that correlate the UAVs track, heading, and winds
//  current_telemetry[i].TAS*cos((current_control[i].MagHeading*(3.14159/180))) + V_w*cos(Chi_w) =
current_telemetry[i].Velocity*cos(current_telemetry[i].Direction)
//  current_telemetry[i].TAS*sin((current_control[i].MagHeading*(3.14159/180))) + V_w*sin(Chi_w) =
current_telemetry[i].Velocity*sin(current_telemetry[i].Direction)


//Wind Finding
		float a = current_telemetry[i].Velocity*cos(current_telemetry[i].Direction) -
current_telemetry[i].TAS*cos((current_control[i].MagHeading*(3.14159/180)));
		float b = current_telemetry[i].Velocity*sin(current_telemetry[i].Direction) -
current_telemetry[i].TAS*sin((current_control[i].MagHeading*(3.14159/180)));

		V_w = sqrt(((a*a) + (b*b)));
		//Chi_w = acos(sqrt(1-((b*b)/((a*a) + (b*b)))));
		Chi_w = atan2(b,a);
		if (Chi_w < 0.0)
		{
			Chi_w_deg = Chi_w * (180/3.14159)+360;
		}
		else
		{
			Chi_w_deg = Chi_w * (180/3.14159);
		}


//New heading and airspeed calculations based off of the above wind values
		float c = current_telemetry[i].Velocity*cos(current_telemetry[i].Direction) - V_w*cos(Chi_w);
```

```
		float d = current_telemetry[i].Velocity*sin(current_telemetry[i].Direction) - V_w*sin(Chi_w);

		V_TASnew = sqrt(((c*c) + (d*d)));
		density = (2 * current_control[i].Pdynamic) / (current_telemetry[i].TAS*current_telemetry[i].TAS);
		Pdynamic_new = 0.5*density*(V_TASnew*V_TASnew);
		Chi_Magnew = atan2(d,c);

		if (current_telemetry[i].Direction*180/3.14159 >0 && current_telemetry[i].Direction*180/3.14159 <= 180)
			{
			Chi_Magnew_deg = Chi_Magnew * (180/3.14159);
			}

		else
			{
				Chi_Magnew_deg = Chi_Magnew * (180/3.14159) + 360;
			}
}

/* //Attempt to send the autopilot new turn rate command based off the new heading desired.
void HeadingAdjust(int i)
{
		//float rate;
		static AutopilotLoopCmd_t turnCom;
		//AutopilotLoopCmd_t turnCom;
		int IDbrent = m_pComm->GetIDFromIndex(i);

  //These divisions were done so as to keep any commanded turn rates less than 20deg/sec
		if (Chi_Magnew_deg - current_control[i].MagHeading > 0  &&  Chi_Magnew_deg - current_control[i].MagHeading <=
20)
		{
		   turnrate = (Chi_Magnew_deg - current_control[i].MagHeading)/1;
		}
		else if (Chi_Magnew_deg - current_control[i].MagHeading > 20  &&  Chi_Magnew_deg - current_control[i].MagHeading
<= 40)
		{
			turnrate = (Chi_Magnew_deg - current_control[i].MagHeading)/2;
	}
		else if (Chi_Magnew_deg - current_control[i].MagHeading > 40  &&  Chi_Magnew_deg - current_control[i].MagHeading
<= 60)
		{
			turnrate = (Chi_Magnew_deg - current_control[i].MagHeading)/3;
		}
		else if (Chi_Magnew_deg - current_control[i].MagHeading > 60  &&  Chi_Magnew_deg - current_control[i].MagHeading
<= 80)
		{
			turnrate = (Chi_Magnew_deg - current_control[i].MagHeading)/4;
		}
		  //  rate = (Chi_Magnew_deg - current_control[i].MagHeading)/1;

			turnCom.Loop=2;				//command a turn rate
			turnCom.Control=1;			 //turn ap_loop_cmd on
			turnCom.Value=turnrate*(3.14159265359/180);  //assign the commanded value
			m_pComm->SendAutopilotLoopControlPacket(IDbrent, &(turnCom));
}*/



//Attempt to send a "return to normal tracking mode" after the turn rate heading was sent
/*void WaypointTrackReturn(int i)
{
		int wayindex;
		static AutopilotLoopCmd_t wayCom;
		int IDbrent4 = m_pComm->GetIDFromIndex(i);
			wayindex = Waypoint_cmd[i];
			wayCom.Loop = 4;
			wayCom.Control = 1;  //Maybe "2"
			wayCom.Value = wayindex;
			m_pComm->SendAutopilotLoopControlPacket(IDbrent4, &(wayCom));//send the command
}
*/
```

```
//Successful command to send the new desired airspeed calculated above
void AirspeedAdjust(int i)
{
	float cmd_speed;
	static AutopilotLoopCmd_t speedCom;
	int IDbrent2 = m_pComm->GetIDFromIndex(i);

		cmd_speed = Pdynamic_new;
		speedCom.Loop = 0;			//command a dynamic pressure
		speedCom.Control = 1;			//turn ap_loop_cmd on
		speedCom.Value = (cmd_speed);		//assign the commanded value
		m_pComm->SendAutopilotLoopControlPacket(IDbrent2, &(speedCom));//send the command
}


//Attempt at ploying the Piccolo into a pure waypoint tracker instead of following straight line tracks between points
/*void WaypointFlyingnotTrackFlying(int i)
{
	int IDbrent7 = m_pComm->GetIDFromIndex(i);
	m_pComm->SendTrackCommandPacket(IDbrent7, Waypoint_cmd[i], true);
}*/


// Trying to calculate then send updating waypoint that is placed at the correct heading to compensate for the wind so as to
// end up at the original desired waypoint...Related to the previous function
/*void WaypointInfoFinding(int i, FPPoint_t& pntWaypoint)//, int next)
{

	int IDbrent5 = m_pComm->GetIDFromIndex(i);
			//AutopilotPkt_t WPPacket;

			//	Waypoint_t origData;
			//	current_waypoint_Latitude  = origData.Lat * (180/3.14159);
			//	current_waypoint_Longitude = origData.Lon * (180/3.14159);
			//	current_waypoint_Altitude  = origData.Alt;
	if(fabs(pntWaypoint.Point.Lat)*180/3.1415926 >0 && fabs(pntWaypoint.Point.Lat)*180/3.1415926 <90)
	{
			current_waypoint_Latitude  = pntWaypoint.Point.Lat  * (180/3.14159);    //TRYING TO
READ OFF WAYPOINT LAT/LONG
			current_waypoint_Longitude = pntWaypoint.Point.Lon  * (180/3.14159);
			current_waypoint_Altitude  = pntWaypoint.Point.Alt;


		 //test
		 //current_waypoint_Latitude  = 39.773098;
		 //current_waypoint_Longitude = -84.111564;
		 //current_waypoint_Altitude  = 350;




			orig     = Waypoint_cmd[i];
			orignext = Waypoint_cmd[i]+1;


	}
}*/


//Attempt to implement the UPDATING "RABBIT" WAYPOINT APPROACH to wind correction
/*void UpdatingWaypointadjustingforWind(int i)
{
		int IDbrent6 = m_pComm->GetIDFromIndex(i);

			ENUCoord WayENU;			//Converting current waypoint LAT/LONG to ENU
			WayENU.lla2enu(current_waypoint_Latitude * 3.1415926/180,
					current_waypoint_Longitude * 3.1415926/180,
					current_waypoint_Altitude,
					Base_X, Base_Y, Base_Z);
```

```
                    e = fabs(current_telemetry[i].East  - WayENU.GetEast());
                        f = fabs(current_telemetry[i].North - WayENU.GetNorth());
                        Dis = sqrt((e*e)+(f*f));
                        float toofar = Dis + 1000;              //This is a distance that the a/c will never reach

    //These are the adjustments to the ENU coords of the a/c to place the new waypoint
                        if (Dis >= 50)
                        {                                       //This is an attempt to place the new waypoint
                                if(Chi_Magnew_deg > 0 && Chi_Magnew_deg <=90)
                                {
                                float angle_deg = Chi_Magnew_deg-90;
                                float angle = angle_deg * (3.1415926/180);      //Check where the datum point is for ENU
                                float abscos = abs(toofar * cos(angle));        //if west and south of wpafb then signs are
okay for the enu99s
                                float abssin = abs(toofar * sin(angle));
                                        enu69east = PosENU.GetEast() + abscos;
                                        enu69north = PosENU.GetNorth() + abssin;
                                }
                                if(Chi_Magnew_deg > 90 && Chi_Magnew_deg <=180)
                                {
                                float angle_deg = Chi_Magnew_deg-90;
                                float angle = angle_deg * (3.1415926/180);
                                float abscos = abs(toofar * cos(angle));
                                float abssin = abs(toofar * sin(angle));
                                        enu69east = PosENU.GetEast() + abscos;
                                        enu69north = PosENU.GetNorth() - abssin;

                                }
                                if(Chi_Magnew_deg > 180 && Chi_Magnew_deg <=270)
                                {
                                float angle_deg = Chi_Magnew_deg-270;
                                float angle = angle_deg * (3.1415926/180);
                                float abscos = abs(toofar * cos(angle));
                                float abssin = abs(toofar * sin(angle));
                                        enu69east = PosENU.GetEast() - abscos;
                                        enu69north = PosENU.GetNorth() - abssin;
                                }
                                if(Chi_Magnew_deg > 270 && Chi_Magnew_deg <=360)
                                {
                                float angle_deg = Chi_Magnew_deg-270;
                                float angle = angle_deg * (3.1415926/180);
                                float abscos = abs(toofar * cos(angle));
                                float abssin = abs(toofar * sin(angle));
                                        enu69east = PosENU.GetEast() - abscos;
                                        enu69north = PosENU.GetNorth() + abssin;
                                }

                                ENUCoord newwpENU;                      //Convert the new waypoint ENU to LLA
                                newwpENU.enu2lla(enu69east, enu69north, WayENU.GetUp(), Base_X, Base_Y,
Base_Z);


                                FPPoint_t newWPInfo;
                                Waypoint_t newlocation;
                        //      AutopilotPkt_t WPPacket;

                        //      newlocation.Lat=newwpENU.GetLat();          //Lat/Long of new waypoint from just
above
                        //      newlocation.Lon=newwpENU.GetLong();         //*180/3.1415926
                        //      newlocation.Alt=newwpENU.GetAlt();

                                newlocation.Lat=39.78*(3.1415926/180);
                                newlocation.Lon=-84.097096*(3.1415926/180);
                                newlocation.Alt=348;


                                FPPoint_t newWPInfo2;
                                Waypoint_t newlocation2;
```

```
                                        newlocation2.Lat=39.775495*(3.1415926/180);
                                        newlocation2.Lon=-84.114660*(3.1415926/180);
                                        newlocation2.Alt=348;
                                        newWPInfo2.Point = newlocation2;
                                        newWPInfo2.Next = 69;
                                        newWPInfo2.PreTurn = 0;
                                        m_pComm->SendWaypointPacket(IDbrent6, &(newWPInfo2), 70);
                                //m_pComm->SendTrackCommandPacket(IDbrent6, 70, true);

                                        newWPInfo.Point = newlocation;          //Trying to send the new waypoint as
waypoint "99"
                                        if (Waypoint_cmd[i] > 0)
                                        {
                                        newWPInfo.Next = 70;
                                        }
                                        newWPInfo.PreTurn = 0;
                                        m_pComm->SendWaypointPacket(IDbrent6, &(newWPInfo), 69);
                                m_pComm->SendTrackCommandPacket(IDbrent6, 69, false);   //send command to head to new
waypoint

        //      third parameter indicates if the vehicle should fly to the waypoint along the
        //  preceding track segment, or if it should go directly to the waypoint, using its
        //      current position as the starting point. Set to TRUE to go directly to the waypoint.

                                }
    //  else
                        //      {
                        //              FPPoint_t origWP;
                        //              Waypoint_t origlocation;
//
                        //              origlocation.Lat = current_waypoint_Latitude;
                        //              origlocation.Lon = current_waypoint_Longitude;
                        //              origlocation.Alt = current_waypoint_Altitude;
//
                        //              origWP.Point = origlocation;
                        //              origWP.Next = Waypoint_cmd[i]+1;
                        //              origWP.PreTurn = 0;
                        //              m_pComm->SendWaypointPacket(IDbrent6, &(origWP), orig);
                        //              m_pComm->SendTrackCommandPacket(IDbrent6, Waypoint_cmd[i],true);
                        //      }
}*/



//WIND CORRECTED SENSOR POINTING APPROACH TO WIND CORRECTION
void SensorAdjustment(int i)
{
        int IDbrent8 = m_pComm->GetIDFromIndex(i);

        //Assume camera is at 45 degree angle off of a/c nose....no gimble


  //Manually input waypoint lats and longs via the "edit" button on Operator Interface
        //They should be:
        //Waypoint 0 --
                point0Lat=39.773292*(3.1415926/180);   //PUT ALL IN RADIANS!!!!!
                point0Lon=-84.099500*(3.1415926/180);
                Alt0=350; // [m]

/*      //Waypoint 0 --
                point0Lat=39.776000*(3.1415926/180);   //FOR THE LOOOONG POINT TO POINT
                point0Lon=-84.117796*(3.1415926/180);
                Alt0=350; // [m]
*/
        //Waypoint 1 --
                point1Lat=39.773530*(3.1415926/180);
                point1Lon=-84.106384*(3.1415926/180);
                Alt1=350; // [m]
```

```
/*		//Waypoint 1 --
			point1Lat=39.776000*(3.1415926/180);	//FOR THE LOOOONG POINT TO POINT
			point1Lon=-84.103704*(3.1415926/180);
			Alt1=350; // [m]
*/
		//Waypoint 2 --
			point2Lat=39.773700*(3.1415926/180);
			point2Lon=-84.111550*(3.1415926/180);
			Alt2=350; // [m]

/*		//Waypoint 2 --
			point2Lat=39.776000*(3.1415926/180);	//FOR THE LOOOONG POINT TO POINT
			point2Lon=-84.090613*(3.1415926/180);
			Alt2=350; // [m]
*/
		//Waypoint 3 --
			point3Lat=39.775525*(3.1415926/180);
			point3Lon=-84.112517*(3.1415926/180);
			Alt3=350; // [m]

		//Waypoint 4 --
			point4Lat=39.777281*(3.1415926/180);
			point4Lon=-84.111355*(3.1415926/180);
			Alt4=350; // [m]

		//Waypoint 5 --
			point5Lat=39.776950*(3.1415926/180);
			point5Lon=-84.099400*(3.1415926/180);
			Alt5=350; // [m]

		//Waypoint 6 --
			point6Lat=39.774950*(3.1415926/180);
			point6Lon=-84.098450*(3.1415926/180);
			Alt6=350; // [m]

		ENUCoord Point0ENU;
		Point0ENU.lla2enu(point0Lat, point0Lon, Alt0, Base_X, Base_Y, Base_Z);

		ENUCoord Point1ENU;
		Point1ENU.lla2enu(point1Lat, point1Lon, Alt1, Base_X, Base_Y, Base_Z);

		ENUCoord Point2ENU;
		Point2ENU.lla2enu(point2Lat, point2Lon, Alt2, Base_X, Base_Y, Base_Z);

		ENUCoord Point3ENU;
		Point3ENU.lla2enu(point3Lat, point3Lon, Alt3, Base_X, Base_Y, Base_Z);

		ENUCoord Point4ENU;
		Point4ENU.lla2enu(point4Lat, point4Lon, Alt4, Base_X, Base_Y, Base_Z);

		ENUCoord Point5ENU;
		Point5ENU.lla2enu(point5Lat, point5Lon, Alt5, Base_X, Base_Y, Base_Z);

		ENUCoord Point6ENU;
		Point6ENU.lla2enu(point6Lat, point6Lon, Alt6, Base_X, Base_Y, Base_Z);



//WAYPOINT 0 CALCULATIONS
		if (Waypoint_cmd[i] == 0  || Waypoint_cmd[i] == 60)
		{
			e2 = fabs(current_telemetry[i].East  - Point0ENU.GetEast());
			f2 = fabs(current_telemetry[i].North - Point0ENU.GetNorth());
			Dis2 = sqrt((e2*e2)+(f2*f2));

			if (Dis2 <= 200)
			//if (Dis2 <= 500)  //Change for looong pt to pt
			{
			theta_one = atan2((Point0ENU.GetNorth()-Point6ENU.GetNorth()),(Point0ENU.GetEast()-
Point6ENU.GetEast()));
```

```
                        //Assume camera is at 45 degree angle off of a/c....no gimble...60 deg FOV
                        LOS_Dis = current_telemetry[i].Altitude / cos((45*(3.1415926/180)));
                        Horiz = sqrt((LOS_Dis*LOS_Dis) - (current_telemetry[i].Altitude*current_telemetry[i].Altitude));

                        //crab angle is difference between ground track and Piccolo's mag heading...not my mag heading new
      Adjust2a = Horiz*sin((current_telemetry[i].Direction*(180/3.1415926)-current_control[i].MagHeading)*(3.1415926/180));

                        star = 3.1415926-1.5708-fabs(theta_one);
                        sinfromNext  = Adjust2a*sin(star);   //-Adjust2a*sin(star);  works fairly well also...not quite sure which is better
                        cosfromNext = Adjust2a*cos(star);

                        enu60east  = Point0ENU.GetEast()+cosfromNext;   //Changed for looong pt to pt...i switched the sin and cos
and then made sin negative
                        enu60north = Point0ENU.GetNorth()+sinfromNext;
                        enu60alt   = Point0ENU.GetUp();

                        // MAJ BLUES WAY
                /*                      theta_m = (90 - current_control[i].MagHeading)*(3.1415926/180);
                                        delta_1 = Horiz*cos(theta_m);
                                        delta_2 = Horiz*sin(theta_m);

                                        enu60east  = Point0ENU.GetEast()-delta_1;
                                        enu60north = Point0ENU.GetNorth()-delta_2;
                                        enu60alt   = Point0ENU.GetUp();
*/
                        ENUCoord newPointENU;
                        newPointENU.enu2lla(enu60east, enu60north, enu60alt, Base_X, Base_Y, Base_Z);

                        FPPoint_t adjWPInfo;
                        Waypoint_t adjWPlocation;

                        adjWPlocation.Lat = newPointENU.GetLat();
                        adjWPlocation.Lon = newPointENU.GetLong();
                        adjWPlocation.Alt = newPointENU.GetAlt();

                        adjWPInfo.Point = adjWPlocation;
                        adjWPInfo.Next = 1;
                        adjWPInfo.PreTurn = 1;

                //              m_pComm->SendWaypointPacket(IDbrent8, &(adjWPInfo), 60);  //If only the initial calculation for the
waypoint is desired
                //              m_pComm->SendTrackCommandPacket(IDbrent8, 60, true);      //i.e. you don't want it to update...use these


                        float e3 = fabs(current_telemetry[i].East  - enu60east);
                        float f3 = fabs(current_telemetry[i].North  - enu60north);
                        float Dis3 = sqrt((e3*e3)+(f3*f3));
                        if (Dis3 >= 100)
                                {
                                m_pComm->SendWaypointPacket(IDbrent8, &(adjWPInfo), 60);
                                m_pComm->SendTrackCommandPacket(IDbrent8, 60, true);
                                }
                        else
                                {
                                m_pComm->SendTrackCommandPacket(IDbrent8, 1, true);
                                }
                        }
                }



//WAYPOINT 1 CALCULATIONS
   else if (Waypoint_cmd[i] == 1  || Waypoint_cmd[i] == 61)
                {
                        e2 = fabs(current_telemetry[i].East  - Point1ENU.GetEast());
                        f2 = fabs(current_telemetry[i].North - Point1ENU.GetNorth());
                        Dis2 = sqrt((e2*e2)+(f2*f2));

                        if (Dis2 <= 350)
```

```
                        //if (Dis2 <= 500) //Change for looong pt to pt
                        {
                        theta_one = atan2((Point1ENU.GetNorth()-Point0ENU.GetNorth()),(Point1ENU.GetEast()-
Point0ENU.GetEast()));


                        //Assume camera is at 45 degree angle off of a/c....no gimble...60 deg FOV
                        LOS_Dis = current_telemetry[i].Altitude / cos((45*(3.1415926/180)));
                        Horiz = sqrt((LOS_Dis*LOS_Dis) - (current_telemetry[i].Altitude*current_telemetry[i].Altitude));

                        //crab angle is difference between ground track and Piccolo's mag heading...not my mag heading new
      Adjust2a = Horiz*sin((current_telemetry[i].Direction*(180/3.1415926)-current_control[i].MagHeading)*(3.1415926/180));

                        star = 3.1415926-1.5708-fabs(theta_one);
                        sinfromNext  = -Adjust2a*sin(star);
                        cosfromNext = Adjust2a*cos(star);

                        enu61east  = Point1ENU.GetEast()+cosfromNext;
                        enu61north = Point1ENU.GetNorth()+sinfromNext;
                        enu61alt   = Point1ENU.GetUp();

                        // MAJ BLUES WAY
/*                              theta_m = (90 - current_control[i].MagHeading)*(3.1415926/180);
                                delta_1 = Horiz*cos(theta_m);
                                delta_2 = Horiz*sin(theta_m);

                                enu61east = Point1ENU.GetEast()-delta_1;
                                enu61north = Point1ENU.GetNorth()-delta_2;
                                enu61alt = Point1ENU.GetUp();
*/
                        ENUCoord newPointENU;
                        newPointENU.enu2lla(enu61east, enu61north, enu61alt, Base_X, Base_Y, Base_Z);

                        FPPoint_t adjWPInfo;
                        Waypoint_t adjWPlocation;

                        adjWPlocation.Lat = newPointENU.GetLat();
                        adjWPlocation.Lon = newPointENU.GetLong();
                        adjWPlocation.Alt = newPointENU.GetAlt();

                        adjWPInfo.Point = adjWPlocation;
                        adjWPInfo.Next = 2;
                        adjWPInfo.PreTurn = 1;

                        float e3 = fabs(current_telemetry[i].East  - enu61east);
                        float f3 = fabs(current_telemetry[i].North  - enu61north);
                        float Dis3 = sqrt((e3*e3)+(f3*f3));

                        if (Dis3 >= 100)
                                {
                                m_pComm->SendWaypointPacket(IDbrent8, &(adjWPInfo), 61);
                                m_pComm->SendTrackCommandPacket(IDbrent8, 61, true);
                                }
                        else
                                {
                                m_pComm->SendTrackCommandPacket(IDbrent8, 2, true);
                                }
                        }
                }



//WAYPOINT 2 CALCULATIONS
                else if (Waypoint_cmd[i] == 2 || Waypoint_cmd[i] == 62)
                {
                        e2 = fabs(current_telemetry[i].East  - Point2ENU.GetEast());
                        f2 = fabs(current_telemetry[i].North - Point2ENU.GetNorth());
                        Dis2 = sqrt((e2*e2)+(f2*f2));

                        if (Dis2 <= 300)
```

```
                    //if (Dis2 <= 500)   //Change for looong pt to pt
                    {
                    theta_one = atan2((Point2ENU.GetNorth()-Point1ENU.GetNorth()),(Point2ENU.GetEast()-
Point1ENU.GetEast()));


                    //Assume camera is at 45 degree angle off of a/c....no gimble...60 deg FOV
                    LOS_Dis = current_telemetry[i].Altitude / cos((45*(3.1415926/180)));
                    Horiz = sqrt((LOS_Dis*LOS_Dis) - (current_telemetry[i].Altitude*current_telemetry[i].Altitude));

                    //crab angle is difference between ground track and Piccolo's mag heading...not my mag heading new
     Adjust2a = Horiz*sin((current_telemetry[i].Direction*(180/3.1415926)-current_control[i].MagHeading)*(3.1415926/180));

                    star = 3.1415926-1.5708-fabs(theta_one);
                    sinfromNext  = -Adjust2a*sin(star);
                    cosfromNext = -Adjust2a*cos(star);      //THIS AND Adjust2a*cos(star) WORK EQUALLY WELL!!!!

                    enu62east  = Point2ENU.GetEast()+cosfromNext;
                    enu62north = Point2ENU.GetNorth()+sinfromNext;
                    enu62alt   = Point2ENU.GetUp();

                                        // MAJ BLUES WAY
/*                          theta_m = (90 - current_control[i].MagHeading)*(3.1415926/180);
                            delta_1 = Horiz*cos(theta_m);
                            delta_2 = Horiz*sin(theta_m);

                            enu62east = Point2ENU.GetEast()-delta_1;
                            enu62north = Point2ENU.GetNorth()-delta_2;
                            enu62alt = Point2ENU.GetUp();
*/

                    ENUCoord newPointENU;
                    newPointENU.enu2lla(enu62east, enu62north, enu62alt, Base_X, Base_Y, Base_Z);

                    FPPoint_t adjWPInfo;
                    Waypoint_t adjWPlocation;

                    adjWPlocation.Lat = newPointENU.GetLat();
                    adjWPlocation.Lon = newPointENU.GetLong();
                    adjWPlocation.Alt = newPointENU.GetAlt();

                    adjWPInfo.Point = adjWPlocation;
                    adjWPInfo.Next = 3;
                    //adjWPInfo.Next = 0;    //Change for loooooong pt to pt
                    adjWPInfo.PreTurn = 1;

          //        m_pComm->SendWaypointPacket(IDbrent8, &(adjWPInfo), 62);
          //        m_pComm->SendTrackCommandPacket(IDbrent8, 62, true);

                    float e3 = fabs(current_telemetry[i].East  - enu62east);
                    float f3 = fabs(current_telemetry[i].North  - enu62north);
                    float Dis3 = sqrt((e3*e3)+(f3*f3));
                    if (Dis3 >= 100)
                              {
                              m_pComm->SendWaypointPacket(IDbrent8, &(adjWPInfo), 62);
                              m_pComm->SendTrackCommandPacket(IDbrent8, 62, true);
                              }
                    else
                              {
                              m_pComm->SendTrackCommandPacket(IDbrent8, 3, true);
                              }
                    }
          }



//WAYPOINT 3 CALCULATIONS
          else if (Waypoint_cmd[i] == 3  ||  Waypoint_cmd[i] == 63)
          {
                    e2 = fabs(current_telemetry[i].East  - Point3ENU.GetEast());
```

```
                    f2 = fabs(current_telemetry[i].North - Point3ENU.GetNorth());
                    Dis2 = sqrt((e2*e2)+(f2*f2));

                    if (Dis2 <= 250)
                    {
                    theta_one = atan2((Point3ENU.GetNorth()-Point2ENU.GetNorth()),(Point3ENU.GetEast()-
Point2ENU.GetEast()));




                    //Assume camera is at 45 degree angle off of a/c....no gimble...60 deg FOV
                    LOS_Dis = current_telemetry[i].Altitude / cos((45*(3.1415926/180)));
                    Horiz = sqrt((LOS_Dis*LOS_Dis) - (current_telemetry[i].Altitude*current_telemetry[i].Altitude));

                    //crab angle is difference between ground track and Piccolo's mag heading...not my mag heading new
      Adjust2a = Horiz*sin((current_telemetry[i].Direction*(180/3.1415926)-current_control[i].MagHeading)*(3.1415926/180));

                    star = 3.1415926-1.5708-fabs(theta_one);
                    sinfromNext  = -Adjust2a*sin(star);
                    cosfromNext = Adjust2a*cos(star);

                    enu63east  = Point3ENU.GetEast()+sinfromNext;
                    enu63north = Point3ENU.GetNorth()+cosfromNext;
                    enu63alt   = Point3ENU.GetUp();

/*                  // MAJ BLUES WAY
                              theta_m = (90 - current_control[i].MagHeading)*(3.1415926/180);
                              delta_1 = Horiz*cos(theta_m);
                              delta_2 = Horiz*sin(theta_m);

                              enu63east = Point3ENU.GetEast()-delta_1;
                              enu63north = Point3ENU.GetNorth()-delta_2;
          */                  enu63alt = Point3ENU.GetUp();


                    ENUCoord newPointENU;
                    newPointENU.enu2lla(enu63east, enu63north, enu63alt, Base_X, Base_Y, Base_Z);

                    FPPoint_t adjWPInfo;
                    Waypoint_t adjWPlocation;

                    adjWPlocation.Lat = newPointENU.GetLat();
                    adjWPlocation.Lon = newPointENU.GetLong();
                    adjWPlocation.Alt = newPointENU.GetAlt();

                    adjWPInfo.Point = adjWPlocation;
                    adjWPInfo.Next = 4;
                    adjWPInfo.PreTurn = 1;

          //        m_pComm->SendWaypointPacket(IDbrent8, &(adjWPInfo), 63);
          //        m_pComm->SendTrackCommandPacket(IDbrent8, 63, true);


                    float e3 = fabs(current_telemetry[i].East  - enu63east);
                    float f3 = fabs(current_telemetry[i].North  - enu63north);
                    float Dis3 = sqrt((e3*e3)+(f3*f3));
                    if (Dis3 >= 150)
                              {
                              m_pComm->SendWaypointPacket(IDbrent8, &(adjWPInfo), 63);
                              m_pComm->SendTrackCommandPacket(IDbrent8, 63, true);
                              }
                    else
                              {
                              m_pComm->SendTrackCommandPacket(IDbrent8, 4, true);
                              }
                    }
          }
```

```
//WAYPOINT 4 CALCULATIONS
        else if (Waypoint_cmd[i] == 4  ||  Waypoint_cmd[i] == 64)
        {
                e2 = fabs(current_telemetry[i].East  - Point4ENU.GetEast());
                f2 = fabs(current_telemetry[i].North - Point4ENU.GetNorth());
                Dis2 = sqrt((e2*e2)+(f2*f2));

                if (Dis2 <= 200)
                {
                theta_one = atan2((Point4ENU.GetNorth()-Point3ENU.GetNorth()),(Point4ENU.GetEast()-
Point3ENU.GetEast()));


                //Assume camera is at 45 degree angle off of a/c....no gimble...60 deg FOV
                LOS_Dis = current_telemetry[i].Altitude / cos((45*(3.1415926/180)));
                Horiz = sqrt((LOS_Dis*LOS_Dis) - (current_telemetry[i].Altitude*current_telemetry[i].Altitude));

                //crab angle is difference between ground track and Piccolo's mag heading...not my mag heading new
    Adjust2a = Horiz*sin((current_telemetry[i].Direction*(180/3.1415926)-current_control[i].MagHeading)*(3.1415926/180));

                star = 3.1415926-1.5708-fabs(theta_one);
                sinfromNext  = Adjust2a*sin(star);
                cosfromNext = -Adjust2a*cos(star);

                enu64east  = Point4ENU.GetEast()+sinfromNext;
                enu64north = Point4ENU.GetNorth()+cosfromNext;
                enu64alt   = Point4ENU.GetUp();

/*                                      // MAJ BLUES WAY
                        theta_m = (90 - current_control[i].MagHeading)*(3.1415926/180);
                        delta_1 = Horiz*cos(theta_m);
                        delta_2 = Horiz*sin(theta_m);

                        enu64east = Point4ENU.GetEast()-delta_1;
                        enu64north = Point4ENU.GetNorth()-delta_2;
                        enu64alt = Point4ENU.GetUp();
*/

                ENUCoord newPointENU;
                newPointENU.enu2lla(enu64east, enu64north, enu64alt, Base_X, Base_Y, Base_Z);

                FPPoint_t adjWPInfo;
                Waypoint_t adjWPlocation;

                adjWPlocation.Lat = newPointENU.GetLat();
                adjWPlocation.Lon = newPointENU.GetLong();
                adjWPlocation.Alt = newPointENU.GetAlt();

                adjWPInfo.Point = adjWPlocation;
                adjWPInfo.Next = 5;
                adjWPInfo.PreTurn = 1;

        //      m_pComm->SendWaypointPacket(IDbrent8, &(adjWPInfo), 64);
        //      m_pComm->SendTrackCommandPacket(IDbrent8, 64, true);

                float e3 = fabs(current_telemetry[i].East  - enu64east);
                float f3 = fabs(current_telemetry[i].North  - enu64north);
                float Dis3 = sqrt((e3*e3)+(f3*f3));
                if (Dis3 >= 100)
                        {
                        m_pComm->SendWaypointPacket(IDbrent8, &(adjWPInfo), 64);
                        m_pComm->SendTrackCommandPacket(IDbrent8, 64, true);
                        }
                else
                        {
                        m_pComm->SendTrackCommandPacket(IDbrent8, 5, true);
                        }
                }
```

```
		}


//WAYPOINT 5 CALCULATIONS
		else if (Waypoint_cmd[i] == 5  ||  Waypoint_cmd[i] == 65)
		{
			e2 = fabs(current_telemetry[i].East  - Point5ENU.GetEast());
			f2 = fabs(current_telemetry[i].North - Point5ENU.GetNorth());
			Dis2 = sqrt((e2*e2)+(f2*f2));

			if (Dis2 <= 600)
			{
			theta_one = atan2((Point5ENU.GetNorth()-Point4ENU.GetNorth()),(Point5ENU.GetEast()-
Point4ENU.GetEast()));

			//Assume camera is at 45 degree angle off of a/c....no gimble...60 deg FOV
			LOS_Dis = current_telemetry[i].Altitude / cos((45*(3.1415926/180)));
			Horiz = sqrt((LOS_Dis*LOS_Dis) - (current_telemetry[i].Altitude*current_telemetry[i].Altitude));

			//crab angle is difference between ground track and Piccolo's mag heading...not my mag heading new
	Adjust2a = Horiz*sin((current_telemetry[i].Direction*(180/3.1415926)-current_control[i].MagHeading)*(3.1415926/180));

			star = 3.1415926-1.5708-fabs(theta_one);
			sinfromNext  = -Adjust2a*sin(star);
			cosfromNext = Adjust2a*cos(star);

			enu65east  = Point5ENU.GetEast()+cosfromNext;
			enu65north = Point5ENU.GetNorth()+sinfromNext;
			enu65alt   = Point5ENU.GetUp();


/*			// MAJ BLUES WAY
				theta_m = (90 - current_control[i].MagHeading)*(3.1415926/180);
				delta_1 = Horiz*cos(theta_m);
				delta_2 = Horiz*sin(theta_m);

				enu65east = Point5ENU.GetEast()-delta_1;
				enu65north = Point5ENU.GetNorth()-delta_2;
				enu65alt = Point5ENU.GetUp();
		*/

			ENUCoord newPointENU;
			newPointENU.enu2lla(enu65east, enu65north, enu65alt, Base_X, Base_Y, Base_Z);

			FPPoint_t adjWPInfo;
			Waypoint_t adjWPlocation;

			adjWPlocation.Lat = newPointENU.GetLat();
			adjWPlocation.Lon = newPointENU.GetLong();
			adjWPlocation.Alt = newPointENU.GetAlt();

			adjWPInfo.Point = adjWPlocation;
			adjWPInfo.Next = 6;
			adjWPInfo.PreTurn = 0;

		//	m_pComm->SendWaypointPacket(IDbrent8, &(adjWPInfo), 65);
		//	m_pComm->SendTrackCommandPacket(IDbrent8, 65, true);

			float e3 = fabs(current_telemetry[i].East  - enu65east);
			float f3 = fabs(current_telemetry[i].North  - enu65north);
			float Dis3 = sqrt((e3*e3)+(f3*f3));
			if (Dis3 >= 100)
				{
				m_pComm->SendWaypointPacket(IDbrent8, &(adjWPInfo), 65);
				m_pComm->SendTrackCommandPacket(IDbrent8, 65, true);
				}
			else
				{
				m_pComm->SendTrackCommandPacket(IDbrent8, 6, true);
				}
```

```
                }
        }


//WAYPOINT 6 CALCULATIONS
        else if (Waypoint_cmd[i] == 6  ||  Waypoint_cmd[i] == 66)
        {
                e2 = fabs(current_telemetry[i].East  - Point6ENU.GetEast());
                f2 = fabs(current_telemetry[i].North - Point6ENU.GetNorth());
                Dis2 = sqrt((e2*e2)+(f2*f2));

                if (Dis2 <= 300)
                {
                theta_one = atan2((Point6ENU.GetNorth()-Point5ENU.GetNorth()),(Point6ENU.GetEast()-
Point5ENU.GetEast()));



/* //THIS WAS AN OLD WAY OF DOING THE CALCULATIONS....Basically it attempted to map the a/c's current position to
where it would be
    //if it were exactly on track....this way placed the new point based on the a/c's location as opposed to placing it based on
            //the location of the current waypoint


        //          double m = fabs(Point6ENU.GetNorth()-Point5ENU.GetNorth());
        //          double n = fabs(Point6ENU.GetEast()-Point5ENU.GetEast());
        //          double Dis_wypts = sqrt((m*m)+(n*n));
        //          T = Dis_wypts - current_telemetry[i].AlongTrack;
        //          double EastonTrack  = Point4ENU.GetEast() + current_telemetry[i].AlongTrack*cos(theta_one);
        //          double NorthonTrack = Point4ENU.GetNorth() + current_telemetry[i].AlongTrack*sin(theta_one);

                if (theta_one*(180/3.1415926)>0 && theta_one*(180/3.1415926)<=90)
                {
                        EastonTrack  = Point5ENU.GetEast() + T*sin(theta_one);
                        NorthonTrack = Point5ENU.GetNorth() + T*cos(theta_one);
                }
                else if (theta_one*(180/3.1415926)>90 && theta_one*(180/3.1415926)<=180)
                {
                        EastonTrack  = Point5ENU.GetEast() + T*cos(theta_one);
                        NorthonTrack = Point5ENU.GetNorth() + T*sin(theta_one);
                }
                else if (theta_one*(180/3.1415926)>-180 && theta_one*(180/3.1415926)<=-90)
                {
                        EastonTrack  = Point5ENU.GetEast() + T*sin(theta_one);
                        NorthonTrack = Point5ENU.GetNorth() + T*cos(theta_one);
                }
                else if (theta_one*(180/3.1415926)>-90 && theta_one*(180/3.1415926)<=0)
                {
                        EastonTrack  = Point5ENU.GetEast() + T*cos(theta_one);
                        NorthonTrack = Point5ENU.GetNorth() + T*sin(theta_one);
                }

                e2onTrack = fabs(EastonTrack  - Point6ENU.GetEast());
                //e2onTrack = fabs(EastonTrack  - current_telemetry[i].East);
                f2onTrack = fabs(NorthonTrack - Point6ENU.GetNorth());
                //f2onTrack = fabs(NorthonTrack - current_telemetry[i].North);
                double Dis_on_Track = sqrt((e2onTrack*e2onTrack)+(f2onTrack*f2onTrack));
                //Dis_on_Track = Horiz2


                //Assume camera is at 45 degree angle off of a/c....no gimble...60 deg FOV
                //float LOS_Dis = current_telemetry[i].Altitude / cos((45*(3.1415926/180)));
                //Horiz = sqrt((LOS_Dis*LOS_Dis) - (current_telemetry[i].Altitude*current_telemetry[i].Altitude));

                //crab angle is difference between ground track and Piccolo's mag heading...not my mag heading new
                if ((current_telemetry[i].Direction*(180/3.1415926)-current_control[i].MagHeading*(3.1415926/180)) >= 0)
                {
                Adjust1a = Dis_on_Track*cos((current_telemetry[i].Direction*(180/3.1415926)-
current_control[i].MagHeading*(3.1415926/180)));
```

```
                        Adjust2a = Dis_on_Track*sin((current_telemetry[i].Direction*(180/3.1415926)-
current_control[i].MagHeading*(3.1415926/180)));
                        }
                        else
                        {
                        Adjust1a = Dis_on_Track*cos((current_telemetry[i].Direction*(180/3.1415926)-
current_control[i].MagHeading*(3.1415926/180)));
                        Adjust2a = Dis_on_Track*-sin((current_telemetry[i].Direction*(180/3.1415926)-
current_control[i].MagHeading*(3.1415926/180)));
                        }
                        enu66east  = Point6ENU.GetEast() + Adjust2a/2;
                        enu66north = Point6ENU.GetNorth() + Adjust1a/2;
                        enu66alt   = Point6ENU.GetUp();
*/

                        //Assume camera is at 45 degree angle off of a/c....no gimble...60 deg FOV
                        LOS_Dis = current_telemetry[i].Altitude / cos((45*(3.1415926/180)));
                        Horiz = sqrt((LOS_Dis*LOS_Dis) - (current_telemetry[i].Altitude*current_telemetry[i].Altitude));

                        //crab angle is difference between ground track and Piccolo's mag heading...not my mag heading new
      Adjust2a = Horiz*sin((current_telemetry[i].Direction*(180/3.1415926)-current_control[i].MagHeading)*(3.1415926/180));

                        star = 3.1415926-1.5708-fabs(theta_one);
                        sinfromNext  = -Adjust2a*sin(star);
                        cosfromNext = -Adjust2a*cos(star);

                        enu66east  = Point6ENU.GetEast()+cosfromNext;
                        enu66north = Point6ENU.GetNorth()+sinfromNext;
                        enu66alt   = Point6ENU.GetUp();

                                        // MAJ BLUES WAY
/*                              theta_m = (90 - current_control[i].MagHeading)*(3.1415926/180);
                                delta_1 = Horiz*cos(theta_m);
                                delta_2 = Horiz*sin(theta_m);

                                enu66east = Point6ENU.GetEast()-delta_1;
                                enu66north = Point6ENU.GetNorth()-delta_2;
                                enu66alt = Point6ENU.GetUp();
*/

                        ENUCoord newPointENU;
                        newPointENU.enu2lla(enu66east, enu66north, enu66alt, Base_X, Base_Y, Base_Z);

                        FPPoint_t adjWPInfo;
                        Waypoint_t adjWPlocation;

                        adjWPlocation.Lat = newPointENU.GetLat();
                        adjWPlocation.Lon = newPointENU.GetLong();
                        adjWPlocation.Alt = newPointENU.GetAlt();

                        adjWPInfo.Point = adjWPlocation;
                        adjWPInfo.Next = 0;
                        adjWPInfo.PreTurn = 0;

                //      m_pComm->SendWaypointPacket(IDbrent8, &(adjWPInfo), 66);
                //      m_pComm->SendTrackCommandPacket(IDbrent8, 66, true);

                        float e3 = fabs(current_telemetry[i].East  - enu66east);
                        float f3 = fabs(current_telemetry[i].North  - enu66north);
                        float Dis3 = sqrt((e3*e3)+(f3*f3));
                        if (Dis3 >= 100)
                                {
                                m_pComm->SendWaypointPacket(IDbrent8, &(adjWPInfo), 66);
                                m_pComm->SendTrackCommandPacket(IDbrent8, 66, true);
                                }
                        else
                                {
                                m_pComm->SendTrackCommandPacket(IDbrent8, 0, true);
                                }
                        }
```

```
		}
}


int main()
{
		//create CCommManager object to communicate with Piccolo
		// 129.92.5.112 is the IP address of the operator interface computer

			m_pComm = new CCommManager(0, 57600, "1.1.1.3", 0);
		//m_pComm = new CCommManager(0, 57600, "129.92.5.112:2000", 0);


		//m_pComm = new CCommManager(1,"",2000);
		//printf("\nHELLO WORLD");
		//print out error and exit if m_pComm doesn't connect
		if(m_pComm->GetLastError() != 0){
		//		printf("\nHELLO WORLD");
				printf("%s", m_pComm->GetLastError());
				printf("\n");
				return 1;
		}

		//set up network callback function
		m_pComm->SetNewNetworkCallBack(NewNetwork, m_pComm);


		pFile1 = fopen ("BrentsLog.txt","w");  //Log file

		//periodic loop to service the communications endpoints
		int i = 0, whosData = 0;

   //Headers for each column in the log file
		fprintf(pFile1," ID   Hours   Minutes   Seconds   Cross Track(m)   Vg      Ground Track    Vtas      Mag Heading
Estimated Wind Vel   Estimated Wind Heading     Crab Angle     Altitude    Mounting Angle  LoS Distance   FootPrint Horizontal
Dis");


		char keypress = 0;
		while(m_pComm && i == 0)
		{
				m_pComm->RunNetwork();
		//			printf("\nHELLO WORLD3");

				LookForAutopilotData(pQ, whosData);


//BRENTS FUNCTION CALLS
	BrentsWindCorrection(i);		//Wind Finding Funciton Call
				//WaypointFlyingnotTrackFlying(i);	//Pure waypoint flying instead of track following function call

	count=count+1;			//Counter so only do this stuff every 15 time hacks.
	  if (count % 15 == 0)
				{
					SensorAdjustment(i);		//Wind Corrected Sensor Pointing function call


				//		UpdatingWaypointadjustingforWind(i);; //Rabbit function
		//		HeadingAdjust(i);		//For turn rate approach
				//		AirspeedAdjust(i);		//For turn rate approach
				}
				//  if (count % 60 == 0)
				//  {				//Trying to manipulate when the function is called so I could send the new heading..
				//  WaypointTrackReturn(i);		//let the a/c adjust...then send it the return to waypoint tracking command
				//  }



		//get commands via keypress
				int rate = 10;
```

```
		if (kbhit()){
			keypress = getch();

			switch(keypress)
			{
				case 'x':
					i = 1;
					printf("\n");
					fclose (pFile1);
					break;
				case 'r': //command a certain turn rate- this was just used as a test
					AutopilotLoopCmd_t loopCom;
					loopCom.Loop = 2;
					loopCom.Control = 1;
					loopCom.Value = (rate*3.14159/180);

					m_pComm->SendAutopilotLoopControlPacket(565, &(loopCom));
					break;
				case '1':
					//print telemetry data for first Network
					whosData = 0;
					break;
				case '2':
					//print telemetry data for second Network
					whosData = 1;
					break;
				case '3':
					//print telemetry data for third Network
					whosData = 2;
					break;
				case '4':
					//print telemetry data for fourth Network
					whosData = 3;
					break;
				case '5':
					//print telemetry data for fifth Network
					whosData = 4;
					break;
				case '6':
					//print telemetry data for sixth Network
					whosData = 5;
					break;
				case '7':
					//print telemetry data for seventh Network
					whosData = 6;
					break;
				case '8':
					//print telemetry data for eighth Network
					whosData = 7;
					break;
				case '9':
					//print telemetry data for ninth Network
					whosData = 8;
					break;
				case '0':
					//print telemetry data for tenth Network
					whosData = 9;
					break;
			}
		}
		//delay to create periodic call, as specified by "Index" in the SDK documentation
		Sleep(10);
	}
	return 0;
}
```

Appendix C: MATLAB Code

SAMPLE MATLAB FOR THE TOP DOWN VIEWS FOR TEST 5 – All tests used the same code simply with different data file calls.

```
%TEST 5 - Adjusted RACETRACK WITH TC=250

clc,close all
clear all

%Analysis of Hardware in the Loop Sim with Flight Test

if exist('Alt0x5Bm0x5D5250') == 0
   load SimTests5datafileE.mat
   disp('File Loading')
end

%Read in Raw flight data from ".mat" file, and build custom Arrays
[Clock] = [Clock0x5Bms0x5D/1000,Day,Hours,Minutes,Seconds];
[Autopilot] = [rad2deg(Lat0x5Brad0x5D),...
   rad2deg(Lon0x5Brad0x5D),...
   Height0x5Bm0x5D*3.281,...
   TAS0x5Bm0x2Fs0x5D*3.281,...
   Direction0x5Brad0x5D,...
   MagHdg0x5Brad0x5D];

[Heading] = [rad2deg(Direction0x5Brad0x5D)];

[Autopilot_Flight] = [Clock,Autopilot];

%Waypoint Locations
WP_latitude = [39.773292; 39.773530; 39.773700; 39.775525;...
   39.777281; 39.776950;39.774950;39.773292];
WP_longitude = [-84.099500; -84.106389; -84.111550;...
   -84.112517; -84.111355; -84.099400;-84.098450;-84.099500];

WP_Altitude = [1148;1148;1148;1148;1148;1148;1148;1148];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
begin = 484;  %Line # in 'Clock' array
end_at = 10802;

% figure('Name',...
%     'HITL Simulation #1: TAS(12kts), Alt(1148ft), Winds(5s/0w m/s)',...
%     'NumberTitle','on')
% hold on
% plot(Autopilot_Flight(begin:end_at,7),Autopilot_Flight(begin:end_at,6),...
%     '--k')
% axis equal
% xlabel ('Longitude (deg)')
% ylabel  ('Latitude (deg)')
% title...
%     ('HITL Autopilot Simulation #1: TAS(12m/s), Alt(1148ft), Winds(5s/0w m/s)')
%
% plot(WP_longitude,WP_latitude,'-ro',...
%               'LineWidth',2,...
%               'MarkerEdgeColor','k',...
%               'MarkerFaceColor',[.49 1 .63],...
%               'MarkerSize',12);
% grid on
% axis equal
% legend({'UAV Flight Path','Desired Waypoints and FlightPath'});
% print -dmeta '1 HITL Autopilot Sim,2D,Actual'
% hold off

%PLOTTING WHERE THE SENSOR WOULD BE
BaseX = 503000;
```

```
BaseY = -4884700;
BaseZ = 4057800;
wp_lla = [deg2rad(WP_latitude) deg2rad(WP_longitude) deg2rad(WP_Altitude)];
lla = [deg2rad(Autopilot_Flight(begin:end_at,6)) deg2rad(Autopilot_Flight(begin:end_at,7))
deg2rad(Autopilot_Flight(begin:end_at,8))];
wyptenu = lla2enu(wp_lla,[BaseX BaseY BaseZ]);
enu     = lla2enu(lla,[BaseX BaseY BaseZ]);

theta = (pi/2) - (Autopilot_Flight(begin:end_at,11));
adjust1=(Autopilot_Flight(begin:end_at,8)./3.281).*sin(theta);  %Only good for 45 degree mounting angle
adjust2=(Autopilot_Flight(begin:end_at,8)./3.281).*cos(theta);

sensorposeast=enu(:,1) + adjust2;
sensorposnorth=enu(:,2)+ adjust1;

figure(11)
hold on
plot(enu(:,1), enu(:,2),'b')
plot(sensorposeast,sensorposnorth,'g')
plot(wyptenu(:,1),wyptenu(:,2),'-ro','LineWidth',2,'MarkerFaceColor',[.49 1 .63], 'MarkerSize',12)
xlabel('East from Datum [m]')
ylabel('North from Datum [m]')
title('Updated UAV & Sensor Tracks (TAS=12m/s, Wind=5 m/s from South)')
legend('UAV Track','Sensor Track','Waypoint',1)
grid on
hold off


%Plot 3D Waypoint Orbit Track
figure1 = figure('Name','HITL Simulation #1: TAS(12m/s), Alt(1148ft), Winds(5s/0w m/s)','NumberTitle','on')
axes1 = axes(...
  'CameraPosition',[-84.13 39.75 2007],...
  'CameraUpVector',[0.1859 0.1775 1.915e+005],...
  'Parent',figure1);
axis(axes1,[-84.12 -84.095 39.77 39.785 800 1500]);
title(axes1,'HITL Autopilot Simulation #1 with Flight Test: TAS(12m/s), Alt(1148ft)');
xlabel(axes1,'Longitude (deg)');
ylabel(axes1,'Latitude (deg)');
zlabel(axes1,'Altitude (ft)');
grid(axes1,'on');
hold(axes1,'all');
plot3(Autopilot_Flight(begin:end_at,7),...  %LONGITUDE  LINES
    Autopilot_Flight(begin:end_at,6),...    %LATITUDE
    Autopilot_Flight(begin:end_at,8),'Parent',axes1);    %ALTITUDE
grid on
hold on
axis equal
plot3(WP_longitude,WP_latitude,WP_Altitude,'-ro',...     %WAYPOINT PLOTS
            'LineWidth',2,...
            'MarkerEdgeColor','k',...
            'MarkerFaceColor',[.49 1 .63],...
            'MarkerSize',12);
axis square
legend1 = legend(axes1,...
{'UAV Flight Path','Desired Waypoints,Flight Path, and Altitude (1148 ft)'},...
'Position',[0.2723 0.3165 0.6554 0.1]);
zlim([800 1500])


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
begin = 12283;  %Line # in 'Clock' array
end_at = 20558;

%2-D PLOT FROM NIDAL
% figure('Name','HITL Simulation #1: TAS(15m/s), Alt(1148ft), Winds(5s/0w m/s)',...
%     'NumberTitle','on')
% plot(Autopilot_Flight(begin:end_at,7),Autopilot_Flight(begin:end_at,6))
```

```
% xlabel ('Longitude (deg)')
% ylabel  ('Latitude (deg)')
% grid on
% axis equal
% hold on
% plot(WP_longitude,WP_latitude,'-ro',...
%               'LineWidth',2,...
%               'MarkerEdgeColor','k',...
%               'MarkerFaceColor',[.49 1 .63],...
%               'MarkerSize',12);
% axis equal
% print -dmeta '4 HITL Autopilot Sim,2D,Conv Lower'


%PLOTTING WHERE THE SENSOR WOULD BE
wp_lla = [deg2rad(WP_latitude) deg2rad(WP_longitude) deg2rad(WP_Altitude)];
lla = [deg2rad(Autopilot_Flight(begin:end_at,6)) deg2rad(Autopilot_Flight(begin:end_at,7))
deg2rad(Autopilot_Flight(begin:end_at,8))];
wyptenu = lla2enu(wp_lla,[BaseX BaseY BaseZ]);
enu     = lla2enu(lla,[BaseX BaseY BaseZ]);

theta = (pi/2) - (Autopilot_Flight(begin:end_at,11));
adjust1=(Autopilot_Flight(begin:end_at,8)./3.281).*sin(theta);  %Only good for 45 degree mounting angle
adjust2=(Autopilot_Flight(begin:end_at,8)./3.281).*cos(theta);

sensorposeast=enu(:,1) + adjust2;
sensorposnorth=enu(:,2)+ adjust1;

figure(12)
hold on
plot(enu(:,1), enu(:,2),'b')
plot(sensorposeast,sensorposnorth,'g')
plot(wyptenu(:,1),wyptenu(:,2),'-ro','LineWidth',2,'MarkerFaceColor',[.49 1 .63], 'MarkerSize',12)
xlabel('East from Datum [m]')
ylabel('North from Datum [m]')
title('Updated UAV & Sensor Tracks (TAS=15m/s, Wind=5 m/s from South)')
legend('UAV Track','Sensor Track','Waypoint',1)
grid on
hold off



% 3-D PLOTTING FROM NIDAL
figure('Name','Simulation #1: TAS(15m/s), Alt(1148ft), Winds(5s/0w m/s)','NumberTitle','on')
plot3(Autopilot_Flight(begin:end_at,7),...
    Autopilot_Flight(begin:end_at,6),...
    Autopilot_Flight(begin:end_at,8));
grid on
hold on
plot3(WP_longitude,WP_latitude,WP_Altitude,'-ro',...
              'LineWidth',2,...
              'MarkerEdgeColor','k',...
              'MarkerFaceColor',[.49 1 .63],...
              'MarkerSize',12);
xlabel ('Longitude (deg)')
ylabel  ('Latitude (deg)')
zlabel ('Altitude (ft)')
zlim([800 1500])


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
begin = 21279;  %Line # in 'Clock' array
end_at = 27453;

%2-D PLOT FROM NIDAL
% figure('Name','HITL Simulation #1: TAS(20m/s), Alt(1148ft), Winds(5s/0w m/s)',...
%     'NumberTitle','on')
```

```
% plot(Autopilot_Flight(begin:end_at,7),Autopilot_Flight(begin:end_at,6))
% xlabel ('Longitude (deg)')
% ylabel  ('Latitude (deg)')
% grid on
% axis equal
% hold on
% plot(WP_longitude,WP_latitude,'-ro',...
%               'LineWidth',2,...
%               'MarkerEdgeColor','k',...
%               'MarkerFaceColor',[.49 1 .63],...
%               'MarkerSize',12);
% axis equal
% print -dmeta '7 HITL Autopilot Sim,2D,TAS Conv Lower'


%PLOTTING WHERE THE SENSOR WOULD BE
wp_lla = [deg2rad(WP_latitude) deg2rad(WP_longitude) deg2rad(WP_Altitude)];
lla = [deg2rad(Autopilot_Flight(begin:end_at,6)) deg2rad(Autopilot_Flight(begin:end_at,7))
deg2rad(Autopilot_Flight(begin:end_at,8))];
wyptenu = lla2enu(wp_lla,[BaseX BaseY BaseZ]);
enu     = lla2enu(lla,[BaseX BaseY BaseZ]);

theta = (pi/2) - (Autopilot_Flight(begin:end_at,11));
adjust1=(Autopilot_Flight(begin:end_at,8)./3.281).*sin(theta);  %Only good for 45 degree mounting angle
adjust2=(Autopilot_Flight(begin:end_at,8)./3.281).*cos(theta);

sensorposeast=enu(:,1) + adjust2;
sensorposnorth=enu(:,2)+ adjust1;

figure(13)
hold on
plot(enu(:,1), enu(:,2),'b')
plot(sensorposeast,sensorposnorth,'g')
plot(wyptenu(:,1),wyptenu(:,2),'-ro','LineWidth',2,'MarkerFaceColor',[.49 1 .63], 'MarkerSize',12)
xlabel('East from Datum [m]')
ylabel('North from Datum [m]')
title('Updated UAV & Sensor Tracks (TAS=20m/s, Wind=5 m/s from South)')
legend('UAV Track','Sensor Track','Waypoint',1)
grid on
hold off



%3-D PLOT FROM NIDAL
figure('Name','HITL Simulation #1: TAS(20m/s), Alt(1148ft), Winds(5s/0w m/s)','NumberTitle','on')
plot3(Autopilot_Flight(begin:end_at,7),...
    Autopilot_Flight(begin:end_at,6),...
    Autopilot_Flight(begin:end_at,8));
grid on
hold on
plot3(WP_longitude,WP_latitude,WP_Altitude,'-ro',...
              'LineWidth',2,...
              'MarkerEdgeColor','k',...
              'MarkerFaceColor',[.49 1 .63],...
              'MarkerSize',12);
xlabel ('Longitude (deg)')
ylabel  ('Latitude (deg)')
zlabel ('Altitude (ft)')
zlim([800 1500])


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
begin = 27974; %Line # in 'Clock' array
end_at = 32413;

%2-D PLOT FROM NIDAL
% figure('Name','HITL Simulation #1: TAS(30m/s), Alt(1148ft), Winds(5s/0w m/s)',...
```

```
%     'NumberTitle','on')
% hold on
% plot(Autopilot_Flight(begin:end_at,7),Autopilot_Flight(begin:end_at,6))
% axis equal
% xlabel ('Longitude (deg)')
% ylabel  ('Latitude (deg)')
% plot(WP_longitude,WP_latitude,'-ro',...
%               'LineWidth',2,...
%               'MarkerEdgeColor','k',...
%               'MarkerFaceColor',[.49 1 .63],...
%               'MarkerSize',12);
% axis equal
% grid on
% print -dmeta '10 HITL Autopilot Sim,2D,TASLow ConvUp,Larg Track'


%PLOTTING WHERE THE SENSOR WOULD BE
wp_lla = [deg2rad(WP_latitude) deg2rad(WP_longitude) deg2rad(WP_Altitude)];
lla = [deg2rad(Autopilot_Flight(begin:end_at,6)) deg2rad(Autopilot_Flight(begin:end_at,7))
deg2rad(Autopilot_Flight(begin:end_at,8))];
wyptenu = lla2enu(wp_lla,[BaseX BaseY BaseZ]);
enu     = lla2enu(lla,[BaseX BaseY BaseZ]);

theta = (pi/2) - (Autopilot_Flight(begin:end_at,11));
adjust1=(Autopilot_Flight(begin:end_at,8)./3.281).*sin(theta);  %Only good for 45 degree mounting angle
adjust2=(Autopilot_Flight(begin:end_at,8)./3.281).*cos(theta);

sensorposeast=enu(:,1) + adjust2;
sensorposnorth=enu(:,2)+ adjust1;

figure(14)
hold on
plot(enu(:,1), enu(:,2),'b')
plot(sensorposeast,sensorposnorth,'g')
plot(wyptenu(:,1),wyptenu(:,2),'-ro','LineWidth',2,'MarkerFaceColor',[.49 1 .63], 'MarkerSize',12)
xlabel('East from Datum [m]')
ylabel('North from Datum [m]')
title('Updated UAV & Sensor Tracks (TAS=30m/s, Wind=5 m/s from South)')
legend('UAV Track','Sensor Track','Waypoint',1)
grid on
hold off


%3-D PLOT FROM NIDAL
figure('Name','HITL Simulation #1: TAS(30m/s), Alt(1148ft), Winds(5s/0w m/s)','NumberTitle','on')
plot3(Autopilot_Flight(begin:end_at,7),...
   Autopilot_Flight(begin:end_at,6),...
   Autopilot_Flight(begin:end_at,8));
grid on
hold on
plot3(WP_longitude,WP_latitude,WP_Altitude,'-ro',...
            'LineWidth',2,...
            'MarkerEdgeColor','k',...
            'MarkerFaceColor',[.49 1 .63],...
            'MarkerSize',12);
xlabel ('Longitude (deg)')
ylabel  ('Latitude (deg)')
zlabel ('Altitude (ft)')
zlim([800 1500])
```

SAMPLE MATLAB FOR THE VARIOUS PARAMETERS AND WIND DATA PLOTS - All tests used the same code simply with different data file calls.

```
%Brent Robinson
%Thesis
%Additional plots for each test

clear all
clc

if exist('data5') == 0
   load SimTests5datafile.mat
   disp('File Loading')
end

begin=36;
end_at=2099;

Hours=data5(begin:end_at,2);
Min=data5(begin:end_at,3);
Sec=data5(begin:end_at,4);

SysTime=(Hours.*3600)+(Min.*60)+Sec;

%SysTime = nameoffile(begin:end_at,);
TAS = data5(begin:end_at,8);
GS = data5(begin:end_at,6);
Alt = data5(begin:end_at,13);
MagHeading = data5(begin:end_at,9);
WindVel = data5(begin:end_at,10);
WindDir = data5(begin:end_at,11);
CT = data5(begin:end_at,5);


figure(1)
%Plot - Velocity vs. time
x=SysTime;
y=TAS;
subplot(4,1,1)
plot(x,y)
xlabel('System Time [s]')
ylabel('TAS [m/s]')
grid on

%Plot - Ground Velocity vs. time
y2=GS;
subplot(4,1,2)
plot(x,y2)
xlabel('System Time [s]')
ylabel('Grnd Spd [m/s]')
grid on

%Plot - Altitude vs. time
y3=Alt;
subplot(4,1,3)
plot(x,y3)
xlabel('System Time [s]')
ylabel('Alt [m]')
grid on

%Plot - Mag Heading vs. time
y3b=MagHeading;
subplot(4,1,4)
plot(x,y3b)
xlabel('System Time [s]')
ylabel('Mag Heading [deg]')
grid on



figure(2)
```

```
%Plot - Wind Velocity vs. time
y4=WindVel;
subplot(3,1,1)
plot(x,y4)
xlabel('System Time [s]')
ylabel('Wind Velocity [m/s]')
grid on

%Plot - Wind Heading vs. time
y5=WindDir;
subplot(3,1,2)
plot(x,y5)
xlabel('System Time [s]')
ylabel('Wind Heading [deg]')
grid on

%Plot - Cross Track Distance vs. time
y6=CT;
subplot(3,1,3)
plot(x,y6)
xlabel('System Time [s]')
ylabel('Cross Track Distance [m]')
grid on


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%

begin=2399;
end_at=4051;

Hours=data5(begin:end_at,2);
Min=data5(begin:end_at,3);
Sec=data5(begin:end_at,4);

SysTime=(Hours.*3600)+(Min.*60)+Sec;

%SysTime = nameoffile(begin:end_at,);
TAS = data5(begin:end_at,8);
GS = data5(begin:end_at,6);
Alt = data5(begin:end_at,13);
MagHeading = data5(begin:end_at,9);
WindVel = data5(begin:end_at,10);
WindDir = data5(begin:end_at,11);
CT = data5(begin:end_at,5);


figure(3)
%Plot - Velocity vs. time
x=SysTime;
y=TAS;
subplot(4,1,1)
plot(x,y)
xlabel('System Time [s]')
ylabel('TAS [m/s]')
grid on

%Plot - Ground Velocity vs. time
y2=GS;
subplot(4,1,2)
plot(x,y2)
xlabel('System Time [s]')
ylabel('Grnd Spd [m/s]')
grid on

%Plot - Altitude vs. time
y3=Alt;
subplot(4,1,3)
```

```
plot(x,y3)
xlabel('System Time [s]')
ylabel('Alt [m]')
grid on

%Plot - Mag Heading vs. time
y3b=MagHeading;
subplot(4,1,4)
plot(x,y3b)
xlabel('System Time [s]')
ylabel('Mag Heading [deg]')
grid on



figure(4)
%Plot - Wind Velocity vs. time
y4=WindVel;
subplot(3,1,1)
plot(x,y4)
xlabel('System Time [s]')
ylabel('Wind Velocity [m/s]')
grid on

%Plot - Wind Heading vs. time
y5=WindDir;
subplot(3,1,2)
plot(x,y5)
xlabel('System Time [s]')
ylabel('Wind Heading [deg]')
grid on

%Plot - Cross Track Distance vs. time
y6=CT;
subplot(3,1,3)
plot(x,y6)
xlabel('System Time [s]')
ylabel('Cross Track Distance [m]')
grid on


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%

begin=4199;
end_at=5431;

Hours=data5(begin:end_at,2);
Min=data5(begin:end_at,3);
Sec=data5(begin:end_at,4);

SysTime=(Hours.*3600)+(Min.*60)+Sec;

%SysTime = nameoffile(begin:end_at,);
TAS = data5(begin:end_at,8);
GS = data5(begin:end_at,6);
Alt = data5(begin:end_at,13);
MagHeading = data5(begin:end_at,9);
WindVel = data5(begin:end_at,10);
WindDir = data5(begin:end_at,11);
CT = data5(begin:end_at,5);


figure(5)
%Plot - Velocity vs. time
x=SysTime;
y=TAS;
subplot(4,1,1)
```

```
plot(x,y)
xlabel('System Time [s]')
ylabel('TAS [m/s]')
grid on

%Plot - Ground Velocity vs. time
y2=GS;
subplot(4,1,2)
plot(x,y2)
xlabel('System Time [s]')
ylabel('Grnd Spd [m/s]')
grid on

%Plot - Altitude vs. time
y3=Alt;
subplot(4,1,3)
plot(x,y3)
xlabel('System Time [s]')
ylabel('Alt [m]')
grid on

%Plot - Mag Heading vs. time
y3b=MagHeading;
subplot(4,1,4)
plot(x,y3b)
xlabel('System Time [s]')
ylabel('Mag Heading [deg]')
grid on



figure(6)
%Plot - Wind Velocity vs. time
y4=WindVel;
subplot(3,1,1)
plot(x,y4)
xlabel('System Time [s]')
ylabel('Wind Velocity [m/s]')
grid on

%Plot - Wind Heading vs. time
y5=WindDir;
subplot(3,1,2)
plot(x,y5)
xlabel('System Time [s]')
ylabel('Wind Heading [deg]')
grid on

%Plot - Cross Track Distance vs. time
y6=CT;
subplot(3,1,3)
plot(x,y6)
xlabel('System Time [s]')
ylabel('Cross Track Distance [m]')
grid on



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%

begin=5539;
end_at=6423;

Hours=data5(begin:end_at,2);
Min=data5(begin:end_at,3);
Sec=data5(begin:end_at,4);
```

```
SysTime=(Hours.*3600)+(Min.*60)+Sec;

%SysTime = nameoffile(begin:end_at,);
TAS = data5(begin:end_at,8);
GS = data5(begin:end_at,6);
Alt = data5(begin:end_at,13);
MagHeading = data5(begin:end_at,9);
WindVel = data5(begin:end_at,10);
WindDir = data5(begin:end_at,11);
CT = data5(begin:end_at,5);

figure(7)
%Plot - Velocity vs. time
x=SysTime;
y=TAS;
subplot(4,1,1)
plot(x,y)
xlabel('System Time [s]')
ylabel('TAS [m/s]')
grid on

%Plot - Ground Velocity vs. time
y2=GS;
subplot(4,1,2)
plot(x,y2)
xlabel('System Time [s]')
ylabel('Grnd Spd [m/s]')
grid on

%Plot - Altitude vs. time
y3=Alt;
subplot(4,1,3)
plot(x,y3)
xlabel('System Time [s]')
ylabel('Alt [m]')
grid on

%Plot - Mag Heading vs. time
y3b=MagHeading;
subplot(4,1,4)
plot(x,y3b)
xlabel('System Time [s]')
ylabel('Mag Heading [deg]')
grid on


figure(8)
%Plot - Wind Velocity vs. time
y4=WindVel;
subplot(3,1,1)
plot(x,y4)
xlabel('System Time [s]')
ylabel('Wind Velocity [m/s]')
grid on

%Plot - Wind Heading vs. time
y5=WindDir;
subplot(3,1,2)
plot(x,y5)
xlabel('System Time [s]')
ylabel('Wind Heading [deg]')
grid on

%Plot - Cross Track Distance vs. time
y6=CT;
subplot(3,1,3)
plot(x,y6)
xlabel('System Time [s]')
ylabel('Cross Track Distance [m]')
```

TASK ID **TASK**

1 Point to Point – Standard Piccolo Wind Correction

FLIGHT PHASE	TASK DESCRIPTION
Wind Estimation	Point to Point track w/ wind corrected sensor pointing code running

FIXED PARAMETERS

GW: 15 lb C.G. pos FS: -- in BL: -- in WL: --	GEAR: N/A Initial Hdg: 0 Crosswind: 0	Initial Position: Straight and level flight

VARIED PARAMETERS

N/A	Weather	Headwind (kts)	N/A	Track Convergence	ALT (m)	Speed (m/s)
0	Wind		--	250	350	20

TEST PROCEDURE

PILOT

1. Maintain straight and level flight

2. Switch into autopilot mode when proper checks are complete. Allow to fly in auotpilot mode for a few minutes to ensure expected performance.

TEST ENGINEER/PILOT NOT FLYING

1. Command the track for two or three laps
2. Run the Wind Finding code only (recording the data).

PILOT	DATE	RUN NUMBER

EVALUATION SEGMENT		
Wind Estimation		

Start Evaluation: Point to Point track and level flight
End Evaluation: Point to Point track and level flight

EVALUATION BASIS

Evaluate the track following characteristics to establish a baseline for a simple point to point waypoint track.

PERFORMANCE STANDARDS	TARGET	DESIRED	ADEQUATE

TASK ID **TASK**

2 Circle Pattern – Standard Piccolo Wind Correction

FLIGHT PHASE	TASK DESCRIPTION
Wind Estimation	Circle pattern w/ normal autopilot settings & wind finding code running

FIXED PARAMETERS

GW: 15 lb C.G. pos FS: -- in BL: -- in WL: --	GEAR: N/A Initial Hdg: 0 Crosswind: 0	Initial Position: Straight and level flight

VARIED PARAMETERS

N/A	Weather	Headwind (kts)	N/A	Track Convergence	ALT (m)	Speed (m/s)
0	Wind		--	250	350	20

TEST PROCEDURE

PILOT

1. Maintain straight and level flight

2. Switch into autopilot mode when proper checks are complete. Allow to fly in auotpilot mode for a few minutes to ensure expected performance.

TEST ENGINEER/PILOT NOT FLYING

1. Command the circular pattern.
2. Run the Wind Finding Code only (recording the data).

PILOT	DATE	RUN NUMBER

EVALUATION SEGMENT
Wind Estimation

Start Evaluation: Circular pattern and level flight
End Evaluation: Circular pattern and level flight

EVALUATION BASIS

Evaluate the precision of the wind estimating code to be compared with the known values gathered from the weather station and the Piccolo telemetry and the simulated tests.

PERFORMANCE STANDARDS	TARGET	DESIRED	ADEQUATE

TASK ID **TASK**

3 Race Track Pattern – Standard Piccolo Wind Correction

FLIGHT PHASE	TASK DESCRIPTION	PILOT	DATE	RUN NUMBER
Wind Estimation	Race Track pattern w/ normal autopilot settings & wind finding code running			

FIXED PARAMETERS

GW: 15 lb C.G. pos FS: -- in BL: -- in WL: --	GEAR: N/A Initial Hdg: 0 Crosswind: 0	Initial Position: Straight and level flight

VARIED PARAMETERS

N/A	Weather	Headwind (kts)	N/A	Track Convergence	ALT (m)	Speed (m/s)
0	Wind		--	250	350	12, 15 20, 30

TEST PROCEDURE

PILOT

1. Maintain straight and level flight

2. Switch into autopilot mode when proper checks are complete. Allow to fly in auotpilot mode for a few minutes to ensure expected performance.

TEST ENGINEER/PILOT NOT FLYING

1. Command the race track pattern at each velocity.
2. Run the Wind Finding Code only (recording the data).

EVALUATION SEGMENT
Wind Estimation

Start Evaluation: Race Track pattern and level flight
End Evaluation: Race Track pattern and level flight

EVALUATION BASIS

Evaluate the precision of the track following capabilities of the Piccolo in a race track pattern in order to establish a baseline.

PERFORMANCE STANDARDS	TARGET	DESIRED	ADEQUATE

TASK ID **TASK**

4 **Race Track Pattern – Varying the Track Convergence Gain**

FLIGHT PHASE	**TASK DESCRIPTION**	**PILOT**	**DATE**	**RUN NUMBER**
Track Convergence Effects	Race Track pattern w/ Varying Track Gains & Wind Finding Code Running			

FIXED PARAMETERS

GW: 15 lb C.G. pos FS: -- in BL: -- in WL: --	GEAR: N/A Initial Hdg: 0 Crosswind: 0	Initial Position: Straight and level flight

VARIED PARAMETERS

N/A	Weather	Headwind (kts)	N/A	Track Convergence	ALT (m)	Speed (m/s)
0	Wind		--	50, 150, 250	350	12, 15 20, 30

TEST PROCEDURE

PILOT

1. Maintain straight and level flight

2. Switch into autopilot mode when proper checks are complete. Allow to fly in auotpilot mode for a few minutes to ensure expected performance.

TEST ENGINEER/PILOT NOT FLYING

1. Command the race track pattern at each velocity.
2. Change the Track Convergence gain after two laps at each velocity
3. Run the Wind Finding Code only (recording the data).

EVALUATION SEGMENT

Wind Estimation and Track Following Evaluation

Start Evaluation: Race Track pattern and level flight
End Evaluation: Race Track pattern and level flight

EVALUATION BASIS

Evaluate the precision of the track following capabilities of the Piccolo in a race track pattern in order to establish a baseline. Vary the track convergence gain to display the increased track following capabilities of the UAV.

PERFORMANCE STANDARDS	**TARGET**	**DESIRED**	**ADEQUATE**

TASK ID **TASK**

5 Race Track Pattern – Adjusting for Sensor Pointing

FLIGHT PHASE	TASK DESCRIPTION	PILOT	DATE	RUN NUMBER
Wind Corrected Sensor Pointing	Race Track pattern w/ Wind Corrected Sensor Pointing Code Running			

FIXED PARAMETERS

GW: 15 lb C.G. pos FS: -- in BL: -- in WL: --	GEAR: N/A Initial Hdg: 0 Crosswind: 0	Initial Position: Straight and level flight

VARIED PARAMETERS

N/A	Weather	Headwind (kts)	N/A	Track Convergence	ALT (m)	Speed (m/s)
0	Wind		--	250	350	12, 15 20, 30

TEST PROCEDURE

PILOT

1. Maintain straight and level flight

2. Switch into autopilot mode when proper checks are complete. Allow to fly in auotpilot mode for a few minutes to ensure expected performance

TEST ENGINEER/PILOT NOT FLYING

1. Command the race track pattern at each velocity.
2. Change the velocity after two laps for each
3. Run the Wind Corrected Sensor Pointing Code (recording the data).

EVALUATION SEGMENT
Wind Correction

Start Evaluation: Race Track pattern and level flight
End Evaluation: Race Track pattern and level flight

EVALUATION BASIS

Evaluate the wind corrected waypoints' ability to adjust for sensor pointing and detail any flight path changes.

PERFORMANCE STANDARDS	TARGET	DESIRED	ADEQUATE

TASK ID **TASK**

6 Point to Point – Wind Correction Sensor Pointing

FLIGHT PHASE	TASK DESCRIPTION	PILOT	DATE	RUN NUMBER
Wind Correction	Point to Point track w/ wind corrected sensor pointing code running			

FIXED PARAMETERS

GW: 15 lb C.G. pos FS: -- in BL: -- in WL: --	GEAR: N/A Initial Hdg: 0 Crosswind: 0	Initial Position: Straight and level flight

VARIED PARAMETERS

N/A	Weather	Headwind (kts)	N/A	Track Convergence	ALT (m)	Speed (m/s)
0	Wind		--	250	350	20

TEST PROCEDURE

PILOT

1. Maintain straight and level flight

2. Switch into autopilot mode when proper checks are complete. Allow to fly in auotpilot mode for a few minutes to ensure expected performance.

TEST ENGINEER/PILOT NOT FLYING

1. Command the track for two or three laps
2. Run the Wind Corrected Sensor Pointing code (recording the data).

EVALUATION SEGMENT		
Wind Correction		

Start Evaluation: Point to Point track and level flight
End Evaluation: Point to Point track and level flight

EVALUATION BASIS

Evaluate the sensor pointing correction results. Compare with those from Test 1 as well as the simulated results.

PERFORMANCE STANDARDS	TARGET	DESIRED	ADEQUATE

Appendix E: Flight Test Results

The following set of flight test results were gathered post-defense in order to obtain initial effects of the wind correction algorithm in a real world situation. Two tests are shown. First, a straight and level flight path and then second a circular orbit. The aircraft was flown in RC mode with the wind finding code running. The results for both tests were disappointing. However, the poor results were not due to the algorithm, but rather a malfunction with the Piccolo II's true airspeed reading on board the aircraft. Due to the inaccurate TAS values, the wind velocity and direction results were completely unreliable. In the first test, the TAS quickly drops to zero and remains there throughout the flight. Obviously the UAV had a positive TAS at all time, thus displaying the error in the Piccolo's readout of the TAS. However, it is interesting to note that the wind estimating algorithm was still operating correctly as the estimated winds were precisely the difference between ground track and flight path. With TAS=0 m/s, the algorithm estimated the wind to be the same as the ground speed, as shown in Figures 111 and 112. The TAS results for the circular orbit test, Figure 113, were non-zero, but still inaccurate and unreliable, producing poor results for the wind estimations found in Figure 114.

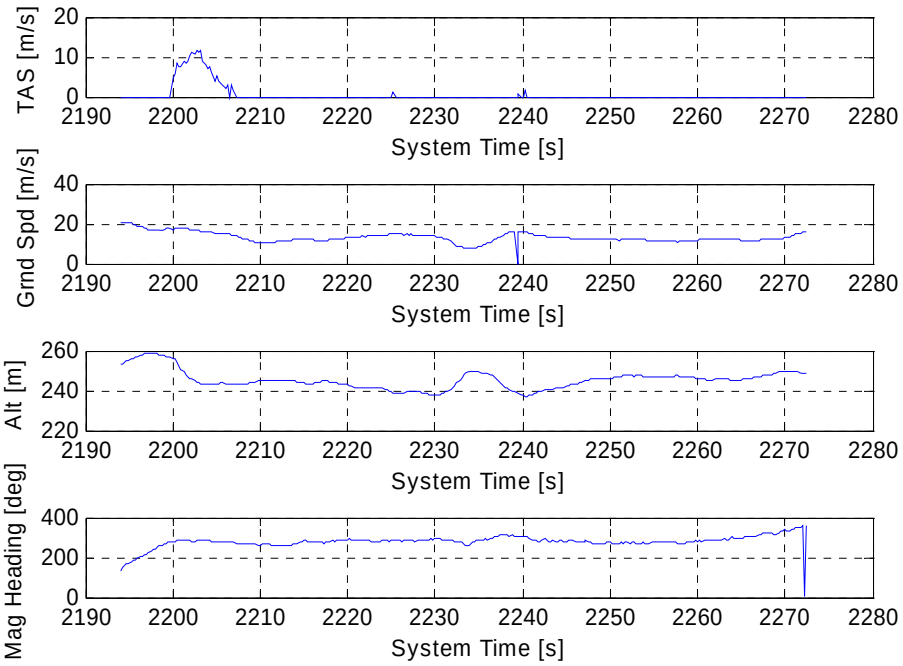

Figure 111. Straight and Level Flight Test Results

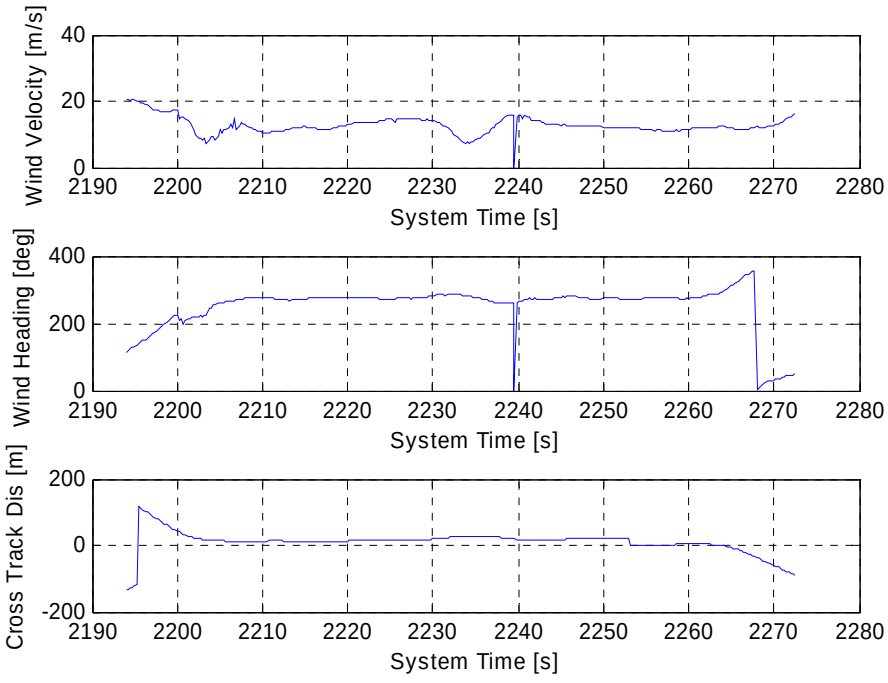

Figure 112. Straight and Level Flight Test Wind Estimations

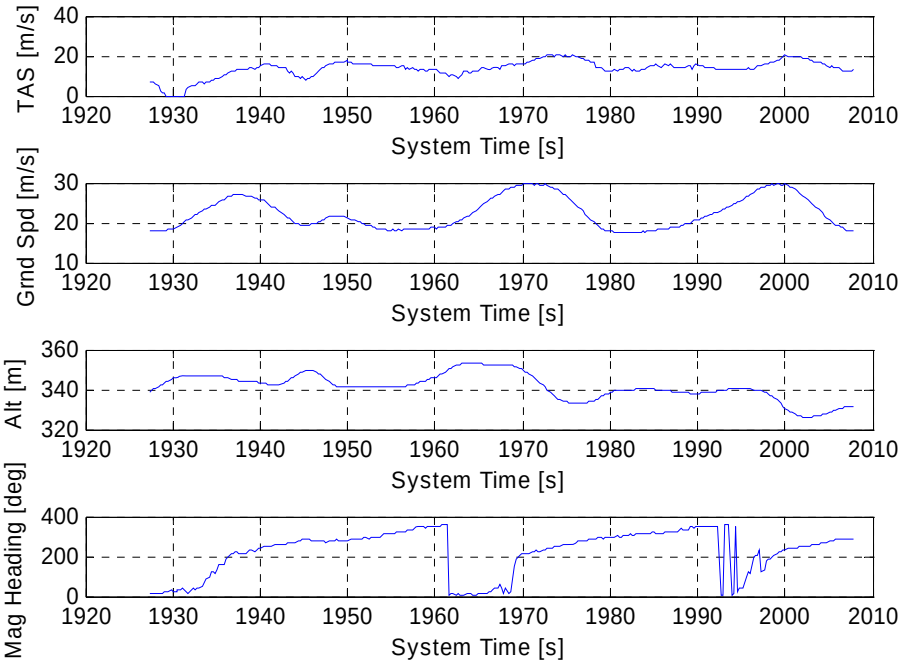

Figure 113. Circular Orbit Flight Test Results

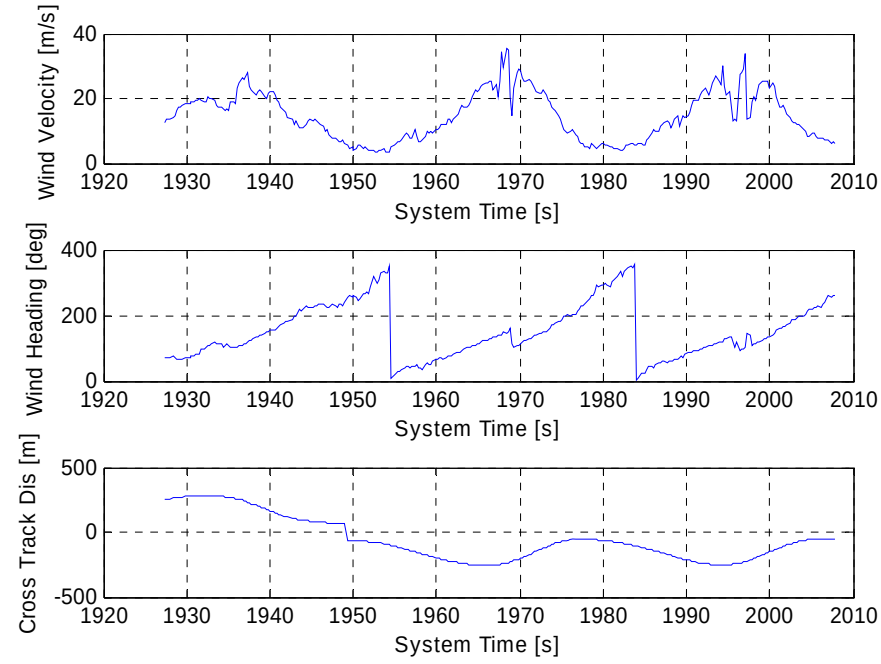

Figure 114. Circular Orbit Flight Test Wind Estimations

Bibliography

"APC 16x8 Pattern Propeller." Retrieved on April 9, 2006 from www2.towerhobbies.com/ cgi-bin/wti0002p?&M=APC. 2006

Bayraktar, S., Fainekos, G.E., Pappas, G.J. *Hybrid Modeling and Experimental Coorperative Control of Multiple Unmanned Aerial Vehicles*. Technical Report. Department of Computer and Information Science, University of Pennsylvannia, PA. December 2004.

Brown, M.J. et. al. "Joint Urban 2003 Street Canyon Experiment." *Joint Urban 2003 Field Study and Urban Mesonets*, Seattle, WA. January 2004.

Bryant, R.L. "Zermello Navigation." Instructor Lecture. Department of Mathematics, Duke University, NC. May 1998.

Bryson, A.E., Ho, Y. *Applied Optimal Control – Optimization, Estimation, and Control*. New York, New York. Hemisphere Publishing Corporation, 1975.

Cionco, R.M., Luces, S.A. "Near Surface Winds from an Enhanced Micro-Mesoscale Simulation System." *The Fifth Conference on Urban Environment*, Vancouver, BC, August 2004.

Dugan, J. *Situational Awareness and Synthetic Vision for Unmanned Aerial Vehicle Flight Testing*. MS Thesis. AFIT/GAE/ENY/06-J2. School of Engineering and Management, Air Force Institute of Technology (AFIT), Wright Patterson AFB, OH. June 2006.

Frew, E., Xiao, X., Spry, S., McGee, T., Kim, Z., Tisdale, J., Sengupta, R., Hendrick, K.J. "Flight Demonstrations of Self-directed Collaborative Navigation of Small Unmanned Aircraft." *Proceedings of the 2004 IEEE Aerospace Conference*, Big Sky, MT, March 2004.

"Futaba - 9CA/CH Computer Systems." Retrieved on April 9, 2006 from http://www.futaba-rc.com/radios/futj85.html. 2006.

Girard, A. R., Hedrick, J.K. "Formation Control of Multiple Vehicles Using Dynamic Surface Control and Hybrid Systems." *International Journal of Control*, 2003. Vol. 76. November 2002.

Jodeh, N. *Development of Autonomous Unmanned Aerial Vehicle Research Platform: Modeling, Simulating, and Flight Testing*. MS Thesis. Department of Aeronautics and Astronautics, Air Force Institute of Technology, OH. March 2006.

King, E. *Distributed Coordination and Control Experiments on a Multi-UAV Testbed.* MS Thesis. Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, MA. September 2004.

Lee, J., Huang, R., Vaughn, A., Xiao, X., Hedrick, J.K., Zennaro, M., Sengupta, R. *Strategies of Path-Planning for a UAV to Track a Ground Vehicle*. Departments of Mechanical and Civil and Environmental Engineering, University of California, Berkeley, CA. May, 2003.

McCarthy, P. *Characterization of UAV Performance and Development of a Formation Flight Controller for Multiple Small UAVs*. MS Thesis. AFIT/GAE/ENY/06-J2. School of Engineering and Management, Air Force Institute of Technology (AFIT), Wright Patterson AFB, OH. June, 2006.

Office of the Secretary of Defense. *Unmanned Aerial Vehicles Roadmap 2002-2027*. Washington: HQ DOD, December, 2002.

"OS 4 Stroke Engines." Retrieved on April 9, 2006 from http://www.ehirobo.com. 2003.

"Smart Digital Magnetometer." Retrieved on April 19, 2006 from http://www.ssec.honeywell.com/magnetic/datasheets/hmr2300.pdf. 2004.

Tin, C. *Robust Multi-UAV Planning in Dynamic and Uncertain Environments*. MS Thesis. Department of Mechanical Engineering, Massachusetts Institute of Technology, MA. September, 2004.

U-Blox AG, Switzerland. Retrieved on December 10, 2005, from http://www.u-blox.com/products/tim_lp.html

Vaglienti, B., Hoag, R., Niculescu, M. *Piccolo System Users Guide.* Hood River OR. Cloud Cap Technology. 18 April 2005.

Vaglienti, B., Niculescu, M. *Hardware in the Loop Simulator for the Piccolo Avionics.* Hood River OR. Cloud Cap Technology. 18 April 200

Vita

Ensign Brent K. Robinson graduated from Palos Verdes Peninsula High School in Rancho Palos Verdes, California. He entered undergraduate studies at the United States Naval Academy in Annapolis, Maryland where he graduated with a Bachelor of Science degree in Aerospace Engineering, with a concentration in Aeronautical Engineering, in May 2005. He was commissioned as an Ensign in the United States Navy on May 27, 2005.

He was directly assigned to Wright Patterson Air Force Base in Dayton, Ohio for graduate studies at the Air Force Institute of Technology. Upon his graduation with a Master's degree in Aeronautical Engineering, he will be assigned to Naval Air Station, Pensacola, Florida for initial pilot training.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
06-13-06	Master's Thesis	June 2005 – June 2006

4. TITLE AND SUBTITLE
An Investigation Into Robust Wind Correction Algorithms for Off-The-Shelf Unmanned Aerial Vehicle Autopilots

5a. CONTRACT NUMBER
5b. GRANT NUMBER
5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Robinson, Brent K., Ensign, USN

5d. PROJECT NUMBER
5e. TASK NUMBER
5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GAE/ENY/06-J14

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFRL/VAA
2210 8TH ST., WPAFB, OH, 45433
Lt Col Lawrence Leny
(937) 255-6500
AFIT Proposal #2003-120, AFIT JON # 05-186

10. SPONSOR/MONITOR'S ACRONYM(S)
11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT The research effort focuses on developing methods to design efficient wind correction algorithms to "piggy-back" on current off-the-shelf Unmanned Aerial Vehicle (UAV) autopilots. Autonomous flight is certainly the near future for the aerospace industry and there exists great interest in defining a system that can guide and control aircraft with high levels of accuracy. The primary systems required to command the vehicles are already in place, but with only moderate abilities to adjust for dynamic environments (i.e. wind effects), if at all. The goal of this research is to develop a systematic procedure for implementing efficient and robust wind effects corrections to existing autopilots. The research will investigate the feasibility of an external dynamic environment control algorithm as a means of improving current, off-the-shelf autopilot technology relating to small UAVs. The research then presents three main focuses. First, a determination of the estimated winds utilizing the existing, on-board sensors. Second, the development of code that incorporates simple mathematical principals to counter the 2-Dimensional wind forces acting on the aircraft; and third, the integration of that code into the on-board navigational system. This "piggy-back" algorithm must assimilate smoothly with the current GPS technologies to provide acceptable and safe flight path following. The design procedures developed were demonstrated in simulation and with flight tests on the SiG Rascal 110 UAV. This report builds the framework from which future wind correction research at AFIT and the ANT Center are based.

15. SUBJECT TERMS
UAV, Autonomous UAV, Wind, Wind Correction, UAV Flight Testing, Piccolo, Piccolo SDK, SIG Rascal

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Paul A. Blue, Maj, USAF AFIT/ENY
U	**U**	**U**	**UU**	182	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-6565 x4714 (paul.blue@afit.edu)

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18